FINAL EVALUATION REPORT
Informix Software, Inc.
INFORMIX-OnLine/Secure

NATIONAL

COMPUTER SECURITY CENTER

9800 Savage Road
Fort George G. Meade
Maryland 20755-6000

21 March 1994

Report No. CSC-EPL-93/004
C-Evaluation Report No.
Library No. S-241,835

This page intentionally left blank

FOREWORD

This publication, the *Final Evaluation Report INFORMIX-OnLine/Secure* is being issued by the National Security Agency under the authority of and in accordance with DoD Directive 5215.1, "Computer Security Evaluation Center." The purpose of this report is to establish the candidate rating for Informix INFORMIX-OnLine/Secure Relational Database Management System. The requirements stated in this report are taken from the *Department of Defense Trusted Database Interpretation of the Trusted Computer System Evaluation Criteria* dated April 1991.

Approved:

Patrick R. Gallagher, Jr. — 21 March 1994
Director,
National Computer Security Center
National Security Agency

ACKNOWLEDGEMENTS

Team Members

Kris Britton
Tammy S. Compton
Christina McBride

National Security Agency
Fort Meade, MD

Cynthia L. Grall

The Aerospace Corporation
El Segundo, CA.

Maria M. King

Institute for Defense Analyses
Alexandria, VA.

TABLE OF CONTENTS

This page intentionally left blank

FIGURES

This page intentionally left blank

TABLES

This page intentionally left blank

EXECUTIVE SUMMARY

The INFORMIX-OnLine/Secure 4.1 relational database management system (RDBMS) executing on AT&T's System V/MLS, Harris Computer Systems' CX/SX, Harris Computer Systems' CX/SX with LAN/SX, or Hewlett Packard's HP-UX BLS trusted products has been evaluated by representatives from the National Security Agency (NSA). In order to establish a rating, the security features of INFORMIX-OnLine/Secure running on these operating systems were examined against the requirements specified by the *Trusted Database Interpretation of the Trusted Computer System Evaluation Criteria* (TDI), dated April 1991.

The trusted operating systems are general-purpose time-sharing systems developed to meet the B1 requirements of the Trusted Computer System Evaluation Criteria (TCSEC). They maintain a security audit trail, provide mandatory access control, and include other security features such as a random password generator, a trusted version of the Bourne Shell, and trusted administrator interfaces.

The evaluation team has determined that, when configured as described in the *INFORMIX-OnLine/Secure Trusted Facility Management* document, the *INFORMIX-OnLine/Secure Administrator's Guide* and the System V/MLS, CX/SX, CX/SX with LAN/SX, or HP-UX BLS trusted management documents, the composite Trusted Computing Base (TCB), consisting of INFORMIX-OnLine/Secure and the operating system, can be configured to satisfy all of the specified requirements in the TDI for classes C2 and B1.

A system that has been rated as being a C2 division system provides a Trusted Computing Base that implements the following:

- User identification and authentication to control general system access,
- Discretionary access control to protect objects and allow users to distribute access to those objects as appropriate,
- Auditing to enforce general user accountability.

In addition to these features, a B1 division system provides mechanisms to enforce a mandatory access control (MAC) policy.

INFORMIX-OnLine/Secure executing on one of the above mentioned operating systems (OS) provides traditional relational database data abstractions (e.g., databases, tables, rows) as well as the capability to store and retrieve multi-media information through these abstractions.

INFORMIX-OnLine/Secure and System V/MLS, CX/SX, CX/SX with LAN/SX, or HP-UX BLS are integrated to provide access control for database objects using three primary mechanisms:

1. All users must identify and authenticate themselves to the composite TCB before any action on the system is permitted. This mechanism is provided by the OS and is relied upon by the RDBMS.
2. INFORMIX-OnLine/Secure provides discretionary access control (DAC) protection to the granularity of a single user for all RDBMS objects, allowing owners of these objects to distribute access to other users.
3. INFORMIX-OnLine/Secure associates sensitivity labels provided by the operating system with all RDBMS objects extending the mandatory access control (MAC) policy enforced by the operating system to these objects.

Objects mediated by INFORMIX-OnLine/Secure are isolated from objects mediated by the OS, requiring that all access to these objects be done through the interface provided by the INFORMIX-OnLine/Secure RDBMS interface.

Chapter 1

Introduction

In April, 1991, the National Security Agency assigned a Vendor Assistance Phase (VAP) evaluation team to prepare Informix to enter the Design Analysis Phase (DAP) with INFORMIX-OnLine/Secure 4.1. Informix entered DAP in January, 1992. This report is the culmination of the evaluation team's analysis of the security features and assurances provided by the composite TCB that consists of the INFORMIX-OnLine/Secure Relational Database Management System (RDBMS) executing as a trusted subject on either AT&T's System V/MLS, Harris Computer Systems' CX/SX, Harris Computer Systems' CX/SX with LAN/SX, or Hewlett Packard's HP-UX BLS operating system. The INFORMIX-OnLine/Secure portion of the TCB includes three modes of operation: a C2 mode and two B1 modes, B1/Enhanced Assurance (B1/EA) and B1/Enhanced Performance (B1/EP). A component (the SQL Engine) is removed from the TCB in the B1/EA configuration in an effort to minimize the TCB. Following is a list of the twelve evaluated configurations:

- AT&T System V/MLS and INFORMIX-OnLine/Secure in the C2 mode, B1/EA mode, or B1/EP mode,
- Harris Computer Systems CX/SX and INFORMIX-OnLine/Secure in the C2 mode, B1/EA mode, or B1/EP mode,
- Harris Computer Systems CX/SX with LAN/SX and INFORMIX-OnLine/Secure in the C2 mode, B1/EA mode, or B1/EP mode,
- Hewlett Packard's HP-UX BLS and INFORMIX-OnLine/Secure in the C2 mode, B1/EA mode, or B1/EP mode.

This report documents the evaluation team's understanding of the system's security design and appraises its functionality and assurance against the *Trusted Database Interpretation of the Trusted Computer Security Evaluation Criteria's* B and C division security requirements. It presents the the team's final analysis of the product design and the results of testing the security features and assurances.

1.1 Evaluation Process Overview

The Department of Defense Computer Security Center was established in January 1981 to encourage the widespread availability of trusted computer systems for use by facilities processing classified or other sensitive information. In August 1985, the name of the organization was changed to the National Computer Security Center. In order to assist in assessing the degree of trust one could place in a given computer system, the DoD Trusted Computer System Evaluation Criteria (TCSEC) was written. The TCSEC establishes specific requirements that a computer system must meet in order to achieve a predefined level of trustworthiness. The TCSEC levels are arranged hierarchically into four major divisions of protection, each with certain security-relevant characteristics. These divisions are, in turn, subdivided into classes. In 1991, the Trusted Database Interpretation of the TCSEC was introduced to be used in conjunction with the TCSEC in applying its requirements to application oriented software systems in general, and database management systems in particular. To determine the division and class at which all requirements are met by a system, the system must be evaluated against the TCSEC (or one of its interpretations) by an NSA, Trusted Product and Network Security evaluation team.

The NSA supports the creation of secure computer products in varying stages of development from initial design to those that are commercially available. Preliminary to an evaluation, products must go through the Proposal Review Phase. This phase includes an assessment of the vendor's capability to create a secure system and complete the evaluation process. To support this assessment, a Preliminary Technical Review (PTR) of the system is done by the NSA. This consists of a quick review of the current state of the system by a small, but expert, team and the creation of a short report on the state of the system. If a vendor passes the Proposal Review Phase, they will enter a support phase preliminary to evaluation. This support phase has two steps: the Vendor Assistance Phase (VAP) and the Design Analysis Phase (DAP). During VAP, the newly assigned team reviews design specifications and answers technical questions that the vendor may have about the ability of the design to meet the requirements. A product will stay in VAP until the vendor's design, design documentation, and other required evidence for the target TCSEC class are complete and the vendor is well into implementation. At that time, the support moves into DAP.

The primary thrust of DAP is an in-depth examination of a manufacturer's design for either a new trusted product or for security enhancements to an existing product. DAP is based on design documentation and information supplied by the industry source. It involves little "hands on" use of the system, but during this phase the vendor should virtually complete implementation of the product. DAP results in the production of an Initial Product Assessment Report (IPAR) by the NSA assessment team. The IPAR documents the team's understanding of the system based on the information presented by the vendor. Because the IPAR contains proprietary information and represents only a preliminary analysis by the NSA, distribution is restricted to the vendor and the NSA.

Products that have completed the support phase with the successful creation of the IPAR enter formal evaluation. Products entering formal evaluation must be complete security systems. In addition, the release being evaluated must not undergo any additional development. The formal evaluation is an analysis of the hardware and software components of a system, all system documentation, and a mapping of the security features and assurances to the TCSEC and its appropriate interpretations. The analysis performed during the formal evaluation requires "hands on" testing (i.e., functional testing and, if applicable, penetration testing). The formal evaluation results in the production of a final report and an Evaluated Products List entry. The final report is a summary of the evaluation and includes the EPL rating which indicates the final class at which the product satisfies all TCSEC requirements in terms of both features and assurances. The final report and EPL entry are made public.

After completion of the Formal evaluation phase, the product enters the rating maintenance phase (RAMP). The rating maintenance phase provides a mechanism to extend the previous rating to a new version of an evaluated computer system product. As enhancements are made to the computer product the ratings maintenance phase ensures that the level of trust is not degraded.

Rating Maintenance is accomplished by using qualified vendor personnel to manage the change process of the rated product during the maintenance cycle. These qualified vendor personnel must have strong technical knowledge of computer security and of their computer product. These trained personnel will oversee the vendor's computer product modification process. They will demonstrate to the Trusted Product and Network Security Evaluation Division that any modification or enhancements applied to the product preserve the security mechanisms and maintain the assurances required by the TCSEC for the rating previously awarded to the evaluated product.

1.2 Informix Software, Inc.

Informix Software, Inc. develops and markets RDBMSs that are designed to be portable to UNIX-like operating systems. Informix was founded in 1979 (as Relational Database Management Systems) with the introduction of the Informix OnLine RDBMS running on UNIX. They entered the security market in 1981 when they released Informix OnLine 3.3 offering discretionary access control security features for RDBMS database and table data abstractions. With the release of INFORMIX-OnLine/Secure 4.1 in 1991, Informix introduced mandatory access control mechanisms into their product and have achieved a rating of C2 and B1 with INFORMIX-OnLine/Secure 4.1 on either the AT&T System V/MLS platform, the Harris Computer Systems' CX/SX platform, the Harris Computer Systems' CX/SX with LAN/SX platform, or the Hewlett Packard HP-UX BLS platform.

1.3 Conventions

The *italic* type style is used for titles of documents when included in the text, and for all command names, instructions, variable and field names, system call names, and data structure names.

The **bold** type style is used for privileges, directory pathnames, system catalog tables, and sensitivity labels.

The terms KB and MB refer to kilobytes and megabytes, respectively.

1.4 Document Organization

This report consists of nine Chapters and three appendices. Chapter 1 is an introduction. Chapter 2 provides an overview of the composite TCB, discussing the responsibilities and roles of each of the TCB components. Chapter 3 discusses the operating system service for requirements and interface specifications for INFORMIX-OnLine/Secure. Chapter 4 presents the product's software architecture. Chapter 5 discusses the product's security architecture in the context of security requirements. Chapter 6 discusses the assurances associated with INFORMIX-OnLine/Secure. Chapter 7 provides the mapping between the C2 requirements specified in the TDI and the C2 configuration features that fulfill those requirements. Chapter 8 provides the mapping between the B1 requirements specified in the TDI and the B1 configuration features that fulfill those requirements. Chapter 9 provides specific evaluator comments concerning INFORMIX-OnLine/Secure. The three appendices contain the evaluated software components, an acronym list, and document references.

This page intentionally left blank

Chapter 2

TCB Overview

The composite TCB consists of INFORMIX-OnLine/Secure 4.1 and the System V/MLS CX/SX, CX/SX with LAN/SX or HP-UX BLS trusted operating system. INFORMIX-OnLine/Secure is not capable of using any of the cross-network facilities of the Harris CX/SX with LAN/SX system. In order for a user on a remote machine to use the RDBMS, that user must start a process on the local machine. Although the TCB is made up of two components, this report focuses on the security architecture and functions of the database portion of the TCB and the relevant interfaces of each OS required to integrate the two components successfully to enforce a full security policy. Figure 2.1 shows the relationship between the RDBMS and its Operating System (OS); the TCB boundary for the composite TCB is delinated with a heavy line. For a complete discussion of the security architecture and design of each OS, the Final Evaluation Reports for each OS should be consulted [[5], [6], [7]].

This chapter generally discusses the following:

- services and functions provided by the INFORMIX-OnLine/Secure portion of the TCB (describing some general relational database management system concepts)
- how the two components integrate to enforce a uniform security policy.
- how INFORMIX-OnLine/Secure integrates with the operating system to form a composite TCB architecture

2.1 INFORMIX-OnLine/Secure Services and Functions

INFORMIX-OnLine/Secure 4.1 is a database management system. A relational database management system is a collection of operating system application programs that facilitate the process of defining, construct-

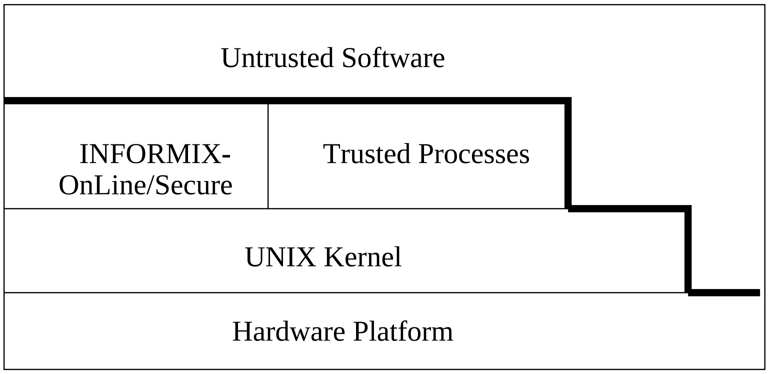


Figure 2.1. The Composite TCB

ing, and manipulating databases for various “real world” applications. A database is a collection of data which is a set of known facts that can be recorded and related in some manner to a given subject.

Defining a database involves specifying the type of data to be stored in the database. Constructing the database is the process of storing the data itself on some storage medium that is controlled by the DBMS. Common uses of databases include such functions as querying the database to retrieve specific data, updating the database to reflect changes in the “real world” being modeled, and generating reports derived from the data.

INFORMIX-OnLine/Secure implements the relational data model[2]. The relational data model represents data in a database as a collection of relations accessed through a query language based on relational calculus. INFORMIX-OnLine/Secure implements relations in a database as tables, which are a collection of data items organized into rows and columns. To create a database is to create a set of tables. Tables are defined by table *schemas* which are templates of the data to be stored in the table the schema defines. The schema contains a set of column descriptions that are associated with a given instance of the subject being modeled. Each column of a table stands for one attribute, characteristic, feature or fact that is true of the topic of that table. Table schemas have information such as the name of the table, column names, data types for column values and constraints that column values must meet (e.g., not NULL, greater than 10, alphanumeric). Tables are populated with rows. Each row of a table stands for one *instance* of the topic of the table and comprises the set of values associated with the columns that are defined in the table schema.

Once a database is opened, database operations are performed on tables. The relational model supports three fundamental operations: *selection*, *projection*, and *join*. The select operation is used to choose a subset of rows in a table. When issuing a select, a user selects rows from a table based on a certain criteria, directing the RDBMS to retrieve only rows that meet that criteria. A projection on a table retrieves certain columns from that table; leaving others aside. This is often used in conjunction with the select operation; selecting specific rows, but retrieving only the values of certain columns of interest from those rows. Joins provide the mechanism for a requester to concatenate related information found in two or more tables. INFORMIX-OnLine/Secure 4.1 provides the ability to manipulate and manage databases and tables through the *Structured Query Language*[8]. All three relational operations are implemented through uses of the SQL **SELECT** statement.

Through its implementation of SQL, INFORMIX-OnLine/Secure implements services (which are common to most RDBMs) to provide the ability to manipulate data in a more flexible and efficient manner. To decrease query searching time, INFORMIX-OnLine/Secure supports the creation of *indexes* on tables which provide a user quicker access to information in the table. INFORMIX-OnLine/Secure supports user-defined *constraints* which allow creators of tables to define the domain or range of data that will be acceptable for a given column. For example, it may be the case that all values of a defined column must be greater than zero or unique in that table. INFORMIX-OnLine/Secure also supports the creation and maintenance of *views* which are logical tables that are derived from other tables. A view does not exist in physical form, but is created through stored SQL statements that perform selections, projections and joins on tables in a database to present a specific view of the database tables. Using this mechanism, users need only invoke a single command to execute multiple SQL statements to retrieve specific information from the database. To support multiple users accessing a table, INFORMIX-OnLine/Secure implements industry standard *isolation levels* which allows database users to determine the degree to which database application programs are isolated from concurrent actions of other programs.

To offer a robust programming interface, INFORMIX-OnLine/Secure supports transaction processing and cursors. A transaction is a collection of one or more SQL statements which is terminated with a *commit* request. Transactions allow users to submit SQL statements to the RDBMS with the guarantee that all statements in the transaction (up to the commit request) will be executed succesfully. If for some reason all statements before the commit request cannot execute to completion (e.g., system crash, semantic error, unavailable resources) the RDBMS will *undo* (termed a *rollback*) all of the statements that were executed in the transaction to restore the database to a consistent state. A cursor is a mechanism to which a program making RDBMS requests can save the results of a query and access at any point without actually making another SQL request. A cursor is created by a program when the program requests the RDBMS to open one. A program requesting a cursor can peruse information placed in the cursor by changing a pointer which points to the current record (called the current row) and executing a *fetch* command which will place the current row in a program variable in the requesting program.

2.2 Composite TCB Policy

The composite TCB security policy requires that no user be allowed to access information in the system unless the user has the requisite authorization. The C2 configuration of the system enforces two primary policies as required by the TCSEC: a discretionary access control (DAC) policy and an individual accountability policy. The system implements four mechanisms to support these policies: identification and authentication, auditing, discretionary access control, and mechanisms to prevent data scavenging (object reuse). The B1 configuration enforces the individual accountability and discretionary access control policies as well as a Mandatory Access Control (MAC) Policy. Seven mechanisms are implemented to support these policies: the same identification and authentication, auditing, discretionary access control and object reuse mechanisms of the C2 configuration along with subject and object labels, mandatory access control, and label integrity mechanisms.

Information protected by the composite TCB is stored in objects. Access to information in these objects is controlled by composite TCB mediation of each subject's request. The subjects controlled by the composite TCB are UNIX processes making requests on behalf of users. The defined set of objects for the composite TCB comprise the objects defined by the operating system and the objects defined by INFORMIX-OnLine/Secure. Given this, the composite TCB interface comprises all the interfaces of the operating system component and the interfaces that INFORMIX-OnLine/Secure provides for objects that it controls. The operating system objects and interfaces are discussed in the appropriate operating system final evaluation report while the INFORMIX-OnLine/Secure objects and interfaces are described in this report on page 73, "RDBMS Protected Resources".

Each TCB subset mediates access to its own objects with the security policy enforced by INFORMIX-OnLine/Secure augmenting and extending the security policy enforced by the operating system. INFORMIX-OnLine/Secure *augments* the operating system DAC policy by applying RDBMS unique discretionary access attributes to RDBMS objects such as Database Administrator (DBA), Resource and Connect for databases and Select, Update and Insert for tables. For a detailed discussion of the INFORMIX-OnLine/Secure DAC policy and mechanisms, see page 75, "Discretionary Access Control". INFORMIX-OnLine/Secure *extends* the OS MAC policy to its objects by maintaining labels for RDBMS objects and relying on the operating system to determine the relationship between any two labels. For a detailed discussion of the INFORMIX-OnLine/Secure MAC enforcement see page 79, "Mandatory Access Control".

2.3 Composite TCB Architecture

The TCB encompasses the totality of protection mechanisms responsible for enforcing a unified security policy over information maintained by the system which comprises two distinct components (TCB subsets): INFORMIX-OnLine/Secure and the operating system. The cooperation of the two TCB subsets ensures the enforcement of the overall security policy. The operating system TCB subset is a B1 UNIX system which provides services to INFORMIX-OnLine/Secure. Specific service interfaces which are used by INFORMIX-OnLine/Secure are discussed in detail on page 12, “Required OS Services”. The INFORMIX-OnLine/Secure TCB subset consists of a number of Unix processes and their associated data structures which are discussed on page 21, “Informix Database Managment System Architecture”.

The two TCB components maintain a hierarchical relationship with the operating system being the most primitive of the TCB subsets.[1] The operating system is responsible for providing services and object abstractions to INFORMIX-OnLine/Secure which are used to protect it from tampering and house its own RDBMS object abstractions.

INFORMIX-OnLine/Secure executes as a set of UNIX processes that works to manage its own data which it stores and retrieves through operating system mechanisms. The evaluated configuration includes three security configurations: the C2, the B1/EA and the B1/EP. All of these configurations maintain the same relationship with the OS. Users have their own instantiation of the RDBMS executing on their behalf. All instances of INFORMIX-OnLine/Secure use a single piece of global shared memory and common disk space to manage and share information. These shared areas are protected from tampering by the use of a special operating system MAC category (on the B1 configurations), **Datahi+IXDATA**, to which no untrusted users have access. Each OS object containing RDBMS information is protected by traditional UNIX protection bits with access granted only to members of the **ix_data** group to which users have no access. All INFORMIX-OnLine/Secure executables are protected from tampering using UNIX protection bits with execute only permission granted to RDBMS users.

INFORMIX-OnLine/Secure security mechanisms are always invoked when accessing INFORMIX-OnLine/Secure objects as these objects are retrieved and stored exclusively through interfaces provided by INFORMIX-OnLine/Secure. This is accomplished by protecting the global shared memory area with the special category and group mentioned above and allowing only the INFORMIX-OnLine/Secure to access this information. To access information, the INFORMIX-OnLine/Secure processes must have the appropriate privilege to bypass the operating system MAC and DAC mechanisms so that they may access INFORMIX-OnLine/Secure data in globally shared areas. INFORMIX-OnLine/Secure does violate the OS security policy but through testing and other supporting arguments, the evaluation team is convinced that it does so in a manner that does not compromise the security of the OS. Different OS's offer different granularities of privileges which give INFORMIX-OnLine/Secure processes the ability to access the data in its shared areas. For a more detailed discussion of the INFORMIX-OnLine/Secure architecture see page 21, “Informix Database Managment System Architecture”. For a discussion of the specific operating system privileges that the RDBMS Kernel uses on the various operating systems in the evaluated configuration see page 11, “Informix in the Operating System Environment”.

[1] TCB subset A is said to be more primitive than TCB subset B if B depends directly on A for services.

2.4 INFORMIX-OnLine/Secure Constraints

INFORMIX-OnLine/Secure uses OS privileges to bypass the OS protection mechanisms. Informix provided and maintains a document titled *Explaining Use of OS Privilege by OnLine/Secure*, henceforth called the constraints document, which describes the constraints under which INFORMIX-OnLine/Secure operates with respect to each operating system included in the evaluated configuration. This document will be updated and maintained for future ports under RAMP. The purpose of the constraints document is to provide a convincing argument that INFORMIX-OnLine/Secure bypasses OS protection mechanisms in a manner that

1. does not violate the integrity of the OS TCB, and
2. does not violate the OS security policy.

This document explicitly lists all OS privileges used by INFORMIX-OnLine/Secure for each OS, explains why each privilege is needed by the RDBMS, and notes all instances where the privilege is invoked. For both the AT&T and Harris operating systems, the only privilege is the root privilege. The constraints document identifies which RDBMS processes execute with the root privilege and why. For HPUX, there is a finer granularity of privileges than just a single root privilege. For this OS, each individual privilege is identified as described above.

In addition, a portion of the constraints document describes the OS components which could be affected by a trusted application executing with those OS privileges used by INFORMIX-OnLine/Secure. Specifically, the constraints document describes the following in detail:

- The three OS TCB protection mechanisms, hardware, MAC, and DAC, and how each mechanism is used by the OS TCB to protect itself.
- The constraints that a trusted application running with OS privilege must adhere to so as not to corrupt the OS component of the TCB. For each constraint presented, an argument is supplied which supports the claim that INFORMIX-OnLine/Secure does not violate the constraint. These constraints and their supporting arguments are presented below.
  - *Modification of the OS TCB executables on disk*: Informix lists the executables that make up each OS TCB. The *INFORMIX-OnLine/Secure Trusted Facility Manual* lists all of the objects created or manipulated by INFORMIX-OnLine/Secure. Informix argues that none of the files that contain OS executables are part of this list.
  - *Modification of each OS kernel and other subjects while executing*: Informix has shown that they understand the architecture of each operating system and understand that a trusted subject must be constrained from writing to **/dev/kmem** and **/dev/mem**. Informix claims that the RDBMS does not open either file, and in fact, neither file is listed in the *INFORMIX-OnLine/Secure Trusted Facility Manual*. Informix also states that the RDBMS does nothing to modify the permissions on either of these files prohibiting an untrusted subject from subsequently modifying them.
  - *Modification of OS TCB protected objects*: Informix has provided a list of OS TCB protected files and discussed the fact that subjects must be constrained from modifying these files, except in a trusted manner. Informix has noted that the only file manipulated by the RDBMS is the OS

audit trail. Detailed design documentation has been provided that discusses how the OS audit trail is manipulated by INFORMIX-OnLine/Secure.

All objects created and manipulated by INFORMIX-OnLine/Secure for untrusted users are owned by the group **ix_data** and, for B1, labeled with the category **IX_DATA**. The group **ix_data** has no members, except root; no untrusted user has the category **IX_DATA** in their category set. The operating system untrusted user objects are thus protected from accidental modification.

Chapter 3

Informix in the Operating System Environment

This chapter discusses INFORMIX-OnLine/Secure in the OS environment. The next section discusses the MAC and DAC relationship between INFORMIX-OnLine/Secure and the OS. On page 12, "Required OS Services" the general services that an OS must provide in order to support INFORMIX-OnLine/Secure are discussed. INFORMIX-OnLine/Secure uses the services of an OS via its interface. Currently, INFORMIX-OnLine/Secure only runs on UNIX-like systems and interfaces with the OS through the UNIX system call mechanism. A number of system calls used to provide the services required by INFORMIX-OnLine/Secure are common to most UNIX-like systems. These system calls are described on page 12, "Required OS Services". The evaluated configuration includes three operating systems:

- AT&T System V/MLS (See page 13, "AT&T System V/MLS and Harris CX/SX Interface")
- Harris CX/SX (See page 13, "AT&T System V/MLS and Harris CX/SX Interface")
- HP-UX BLS (See page 16, "HP-UX BLS")

3.1 OS Administrative Issues

In order to access INFORMIX-OnLine/Secure, users must be a member of the UNIX group **ix_users**. In the B1 configurations, two sensitivity labels must also be defined: **Datahi** and **Datalo**. **Datahi** must be dominated by the operating system's SYSTEM HIGH; **Datalo** must dominate the operating system's SYSTEM LOW. A regular user cannot initiate a database session if any of the following are true:

- the sensitivity label of the user strictly dominates **Datahi**
- the sensitivity label of the user is strictly dominated by **Datalo**
- the sensitivity label of the user is incompatible with the sensitivity labels between **Datalo** and **Datahi**

Two additional UNIX groups are needed by INFORMIX-OnLine/Secure: **ix_dbsso** and **ix_dbsa**. The administrative accounts for the Database System Security Officer (DBSSO) and Database System Administrator (DBSA) must be a member of the respective group. In the B1 configurations, the categories **IX_DBSSO** and **IX_DBSA** are also required and must be in the category set of the DBSSO and DBSA, respectively.

INFORMIX-OnLine/Secure uses UNIX file system objects, shared memory, semaphores, and device objects. For example, a file or raw disk partition is used by INFORMIX-OnLine/Secure to contain the databases. To protect these objects, INFORMIX-OnLine/Secure requires that the UNIX group **ix_data** exist at system initialization. No users, with the exception of root, are a member of this group. Semaphore sets and shared memory are owned by the group **ix_data**. All files are owned by the DBSA account. All UNIX objects have permissions 660 with the exception of semaphores which are read and alter (not write) by owner and group. For the B1 configurations, all such UNIX objects are labeled **Datahi + IX_DATA**. No user has the category **IX_DATA** in their category set. Lastly, in the B1 configurations, INFORMIX-OnLine/Secure executables are labeled at the operating system's SYSTEM LOW in the AT&T System V/MLS and Harris systems and it is recommended in the Informix **Trusted Facility Manual** that a "SYSTEM LOW" label be created in the Hewlett Packard HP-UX BLS product to protect INFORMIX-OnLine/Secure executables. No user must be allowed to initiate a database session at SYSTEM LOW.

3.2 Required OS Services

INFORMIX-OnLine/Secure requires the following six types of services from the OS:

1. device processing requests
2. file processing requests
3. shared memory requests
4. security requests
5. process requests
6. semaphore requests

Device processing requests are needed by INFORMIX-OnLine/Secure to open, close, read, and write devices. These capabilities are required so that INFORMIX-OnLine/Secure can manipulate the raw devices. INFORMIX-OnLine/Secure uses raw devices when a chunk is defined to be a device or a part of a device.

File processing requests provide the means by which data contained in a file can be manipulated. INFORMIX-OnLine/Secure uses file processing requests when a chunk is defined to be a file rather than a device (see page 33, "Chunk Management"). The **tbconfig** file maintains configuration information for INFORMIX-OnLine/Secure and is used to determine the size of the raw device and the location of global shared memory. The RDBMS Kernel (Relational Storage Access Methods (RSAM), the transient and support processes) reads the **tbconfig** file and writes it out to **/tmp/tbconfig**[1]. In addition, **tbload** manipulates two files, **label.lok** and **label.map**, when importing data that contains undefined or incompatibly defined labels (see page 60, "Support and Transient Processes"). The support processes require file manipulation services so that *stbinit* can write into a message log and a console log whose names are specified in **tbconfig**; *stbcheck* can create temporary files; and *stbstat* can create files to hold monitoring information. The Secure Administrator Front End (SAFE) creates a file to hold an audit mask report and the SQL Engine reads the **mlsconfig** file. Lastly, *datextract* creates a file to hold the audit data extracted from the OS audit log (see page 90, "Audit Analysis Officer").

Shared memory requests are needed so that the RDBMS Kernel may establish interprocess communication through the use of shared memory. The needed functionality includes creating a shared memory segment, attaching to shared memory, and detaching from shared memory. Shared memory is used for two purposes. In INFORMIX-OnLine/Secure, a large segment of main memory is shared among multiple RDBMS Kernel processes and accessible to every active RDBMS Kernel process. Using a shared memory scheme reduces disk I/O since data and index pages are buffered on a system-wide basis rather than a per-process basis. This memory is called *global shared memory* and is described in detail on page 40, "Global Shared Memory".

[1] /tmp/tbconfig is owned by the DBSA account and has permissions 660. In the B1 configurations, this file is labeled **Datahi + IX_DATA**.

Shared memory is also used in the EA configuration for communication between SQL Engine and the RSAM process. This shared memory is called *session shared memory* and is discussed on page 59, "Session Startup".

In the area of *security requests*, INFORMIX-OnLine/Secure relies on the OS to control users' attempts to access INFORMIX-OnLine/Secure data. Since INFORMIX-OnLine/Secure has no separate login procedure, it relies on the OS for user information such as user ID, group ID, and sensitivity label. In the B1 configurations, INFORMIX-OnLine/Secure also uses the services of the OS to retrieve the sensitivity labels on files, perform sensitivity label comparisons, convert sensitivity labels to other formats, and modify session sensitivity labels, which can be done by privileged users. Lastly, INFORMIX-OnLine/Secure must be able to send audit records to the OS so that audit information can be included in the audit trail (See page 85, "Auditing" and page 90, "Audit Analysis Officer").

Process requests provide INFORMIX-OnLine/Secure with a means to create, execute, and terminate OS processes; use simple interprocess communication mechanisms, such as pipes; and determine the status of a process. INFORMIX-OnLine/Secure uses process requests to start the SQL Engine and the RSAM process on behalf of a user and to terminate these processes when they have completed their task or when error conditions arise.

INFORMIX-OnLine/Secure uses simple interprocess communication mechanisms such as signals and pipes. Signals are used, for example, when a dbspace (See page 31, "Disk Management and Structures" for a discussion of dbspaces) is being dropped and all processes affected need to be notified. Pipes are used to transfer information between two processes. INFORMIX-OnLine/Secure uses pipes for communication between the transient and its support process; for synchronization between the SQL Engine and RSAM in the EA configuration; and between a client process and the SQL Engine[2]. Sometimes it is necessary to determine the status of a process in terms of whether it has died, is still executing, or has terminated with an exit status. INFORMIX-OnLine/Secure requires these services from the OS.

Semaphore requests are used when RSAM must wait on something. One semaphore is available per RSAM process.

A number of system calls used to provide the services required by INFORMIX-OnLine/Secure are common to most UNIX-like systems. These system calls are listed on Table 3.1.

3.3 AT&T System V/MLS and Harris CX/SX Interface

System V/MLS is a product of AT&T. It is a B1 operating system based on an enhanced version of UNIX System V. System V/MLS is compatible with UNIX systems and offers a discretionary access control mechanism, mandatory access control mechanism, and an auditing mechanism. In addition, a variety of other security features are available including a random password generator, labeling of printer output, and a trusted shell. See the *AT&T System V/MLS Final Report* for more details about this system.

CX/SX is a product of Harris Computer Systems. The CX/SX operating system is a B1 operating system based on AT&T System V/MLS and includes features from Berkeley Software Distribution (BSD) UNIX

[2] Front-end tools are used to communicate between SQL and the user. These take the form of 1) an embedded SQL program which has C, Fortran, or Cobol as the host language; 2) an Informix/4GL program; 3) an interactive interface, such as ISQL or dbaccess; 4) a third party tool which is compatible with the UFE/SQL Engine protocol; or 5) a user-developed tool if it is compatible with the UFE/SQL Engine protocol.

Device and File Processing Requests	
access	Determines access based on real user and group IDs. Used to determine if a file exists.
close	Close raw devices or files.
creat	Creat a file.
dup,dup2	In the EA configuration, duplicates open file descriptors between the SQL Engine and RSAM.
fcntl	Provides control for open files.
ioctl	Provides functions for character special devices.
link	Link to a file. Used to move a file.
lseek	Move the read/write file pointer when reading/writing a raw device or file.
mkdir	Create directories in the hierarchy for files.
open	Open raw devices or files.
read,write	Read/write raw devices or files.
rename	Rename raw devices.
stat,fstat	Get status information about a raw device or file.
unlink	Remove a file.
Shared Memory Requests	
shmat	Attach to shared memory.
shmctl	Provides control functions for accessing shared memory.
shmdt	Detach from shared memory.
shmget	Create shared memory.
Security Requests	
chdir	Change working directory.
chmod	Modify permission bits.
chown	Change owner and group of a file.
gethostname	Get name of host for future reference. Not subsequently referenced in the evaluated configuration.
getpid,getppid	Get process ID, parent process ID.
getuid,getgid	Get user ID and group ID.
setsid	Set session ID for *tbinit* after DBSA logs off.
umask	Used by transient to set correct permissions when copying **tbconfig** for support process.
uname	Get name of current UNIX system.
Process Requests	
execve	Executes a file in a new process.
_exit	Termination of a process.
fork,vfork	Create a new process.
kill	Used to send a signal to another process.
pipe	Interprocess communication mechanism used to synchronize access to local shared memory in EA configuration.
sigblock	Used to block certain signals, primarily in the audit analysis program.
sigsetmask	Set the current signal mask.
sigmask	Used to construct a signal mask.
wait	Wait for a child process to terminate.
Semaphore Requests	
semctl	Provides semaphore control operations.
semget	Used to get a set of semaphores.
semop	Operations for manipulating semaphores.

Table 3.1. System Calls Common to UNIX-like Systems

and SunOS[TM]. CX/SX runs on Harris' Night Hawk 4000 series machines, which contain up to 8 RISC-based processors. CX/SX has a multi-threaded kernel to take advantage of these multiple processors. CX/SX includes DAC, MAC, auditing, and trusted path security features. See the *CX/SX Final Report* for more details about this system.

CX/SX and System V/MLS provide identical interfaces and are used in exactly the same way by INFORMIX-OnLine/Secure. The remainder of this section discusses the interface to both systems.

Objects are marked with a user ID that indicates the owner of the object, a group ID which indicates the group owning the object, and a sensitivity label indicating the sensitivity of the information contained in the object. The combination of the group ID and the sensitivity label is called a *privilege*. This concept of privilege means something very different than it does in UNIX systems based on Secureware, such as HP-UX BLS (see page 16, "HP-UX BLS"). At any point during a login session, a user is associated with a particular privilege. The system uses the user's privilege and the user ID, group ID and sensitivity label of the object to determine the subject's discretionary and mandatory access to the object.

The only special user is the *super-user*. A process executing on behalf of the super-user is said to be running with *root* permission. A process executing with root permission can bypass all security mechanisms. In INFORMIX-OnLine/Secure, RSAM and the transient processes execute with root permission. This section describes the specific interfaces used by INFORMIX-OnLine/Secure and discusses how INFORMIX-OnLine/Secure uses the root permission.

3.3.1 The MLS Library

The *libmls* routines are provided to isolate applications and the Operating System Kernel from details of the sensitivity label implementation. INFORMIX-OnLine/Secure uses two routines from this library: *mls_dom* and *makecf*.

As previously stated, the combination of group ID and sensitivity label is called a privilege. The group ID, in addition to designating group membership, is used to index into a data structure specifying the sensitivity label to be associated with that group. INFORMIX-OnLine/Secure uses the *mls_dom* interface by providing the pertinent group IDs. This routine uses the group IDs as an index to retrieve and then compare the two sensitivity labels. First, INFORMIX-OnLine/Secure makes a simple comparison of the two sensitivity labels. If they are equal, INFORMIX-OnLine/Secure does not invoke the OS interface. If they are not equal, INFORMIX-OnLine/Secure invokes the *mls_dom* routine. A "1" is returned to INFORMIX-OnLine/Secure if the sensitivity label referenced by the first group ID dominates that referenced by the second. Otherwise, a "0" is returned.

In general, programs represent sensitivity labels in canonical form while hardcopy and terminal output require human-readable markings. The *makecf* interface provided by the OS is used by INFORMIX-OnLine/Secure to convert a sensitivity label in human-readable form to one in canonical form.

3.3.2 The Labels File Manipulation Library

The **/mls/labels** file contains the list of sensitivity labels assigned by the OS security administrator. The INFORMIX-OnLine/Secure DBSSO must communicate with the OS security administrator to correctly

[TM] SunOS is a trademark of Sun Microsystems, Inc.

establish sensitivity labels for the RDBMS. All labels defined for use on the system are contained in this file. The file contains a set of structures, one for each defined sensitivity label. Each structure contains the group ID used to point to the given label, the discretionary group ID, some associated flags, the canonical form of the sensitivity label, and some other information. The *getlblent* library contains routines for manipulating the **/mls/labels** file. INFORMIX-OnLine/Secure uses the following routines: *getlblgid*, *getlblmatch*, and *setlblent*. All routines that manipulate the **/mls/labels** file require root permission.

INFORMIX-OnLine/Secure provides a group ID to the *getlblgid* routine. The **/mls/labels** file is searched until a structure is found that contains the desired group ID. A pointer to this structure is returned to INFORMIX-OnLine/Secure. The *getlblmatch* routine is used by INFORMIX-OnLine/Secure to find all structures in the **/mls/labels** file that contain values which match the nonnegative integer and non-NULL character fields of the structure provided by INFORMIX-OnLine/Secure. In either case, if there is no such structure, a NULL pointer is returned. The *setlblent* routine is used by INFORMIX-OnLine/Secure to effectively rewind the label file to allow repeated searches.

3.3.3 Other Specific Interfaces

Two functions are provided to open and write to the audit device, called the Security Audit Trail device (also called trace devices): *tr_open* and *twrite*. INFORMIX-OnLine/Secure uses *tr_open* to open this audit device and gets a file descriptor if the call is successful. If the audit device is busy, *tr_open* will either fail or wait until the device becomes available. The operating system does not keep the audit device open; it is opened as needed by the trusted processes. Subsequently, INFORMIX-OnLine/Secure can use this file descriptor to write to the audit device via the *twrite* function. Both *tr_open* and *twrite* require root permission. In addition, the *datextract* audit analysis tool uses the *satfmt* system call to format the audit trail file. This system call requires root permission.

INFORMIX-OnLine/Secure also uses the *setgid* and *getgid* system calls to set and retrieve the real and effective group ID. A transient process uses the *setgid* system call to set the privilege of the child process that it forked to **Datahi + IX_DATA**. Once the privilege is set, the child process does an *exec* to the appropriate support process. In order to set the privilege, the transient process must execute with the root permission.

INFORMIX-OnLine/Secure uses the *getgrnam* system call to retrieve selected information from the **/etc/group** file by providing a group name. This system call returns a pointer to a structure containing relevant information such as the numerical group ID, encrypted password, and member names for the provided group name. If no such name is found a NULL pointer is returned. The transient process uses the *getgrnam* system call to convert the string "Datahi + IX_DATA" to its numerical group ID. The *setgid* system call takes the numerical group ID as an argument rather than the string.

3.4 HP-UX BLS

HP-UX BLS is a product of the Hewlett Packard Corporation. HP-UX BLS is a B1 operating system compatible with AT&T System V Interface Definition 2 and IEEE POSIX 1003.1 and 1003.2 standards but also includes important features from BSD 4.3. HP chose to incorporate a standard trusted system technology developed by SecureWare, an independent software firm, which includes multilevel access control, as well as identification, authorization and accountability protected subsystems, and a trusted interface for use by system administrators. HP-UX BLS runs on HP's proprietary HP9000/S800 family of computers. See the *HP-UX BLS Final Report* for more details about this system.

In HP-UX BLS, objects are marked with a user ID, group ID, and sensitivity label. However, the concept of privilege means something very different than it does in UNIX systems based on AT&T System V/MLS. Privileges refer to special permissions that allow a process to perform certain actions, in some cases not allowed to regular users. INFORMIX-OnLine/Secure uses six privileges when running on HP-UX BLS: **allowdacaccess**, **allowmacaccess**, **writeaudit**, **setprocident**, **owner**, and **multileveldir**. The **allowdacaccess** privilege allows a process executing with this privilege unconditional discretionary access to all objects. The **allowmacaccess** allows a process executing with this privilege unconditional mandatory access to all objects. The **writeaudit** privilege allows a process executing with this privilege to write to the audit device. The **setprocident** privilege allows a process unrestricted use of setuid and setgid system calls. The **owner** privilege allows a process to bypass owner checks on files and IPC objects. Lastly, the **multileveldir** privilege results in a process not being diverted to the individual subdirectories that comprise a multilevel directory. Table 3.2 lists all the privileges, which INFORMIX-OnLine/Secure programs use the privilege, and why. This section describes the specific HP-UX BLS interfaces used by INFORMIX-OnLine/Secure and discusses the details of how INFORMIX-OnLine/Secure uses these privileges.

3.4.1 The MAC Library

In HP-UX BLS, security information for subjects and objects is stored in a data structure called a *tag pool.* A subject, or process, has three entries, or tags, in its tag pool: an ACL tag, a sensitivity label tag representing the current sensitivity label of the process, and a clearance tag representing the maximum sensitivity label of the process. An object has two entries in its tag pool: an ACL tag and a sensitivity label tag.

A sensitivity label comprises a hierarchical classification name and a set of zero or more non-hierarchical category names. A classification has a human-readable form appropriate for printing, called the external representation (er), a numerical representation stored in the tag pool called a tag, and a structure for internal use, called the internal representation (ir). The classification database stores the mappings of the numerical and external representations for classifications while the category database stores these mappings for categories. A synonym database is also maintained by the OS administrator which stores abbreviations for external representations of classifications, categories, and sensitivity labels.

INFORMIX-OnLine/Secure only stores the tag form of the sensitivity label. However, it is not possible to go directly from the tag form to the er form of a sensitivity label, and vice versa. Instead, the tag must first be converted from tag from to ir form, and then to er form. HP-UX BLS provides the *mand* library routines to manipulate er, ir, and tag forms of classifications, categories, and sensitivity labels. INFORMIX-OnLine/Secure uses the following routines from the *mand* library: *mand_er_to_ir*, *mand_ir_to_er*, *mand_ir_to_tag*, *mand_tag_to_ir*, *mand_init*, *mand_end*, *mand_alloc_ir*, *mand_free_ir*, *mand_ir_relationship*, and *mand_tag_relationship*.

The routine *mand_ir_to_tag* allows INFORMIX-OnLine/Secure to initialize a tag from the ir of a sensitivity label. The *mand_tag_relationship* routine can then be used by INFORMIX-OnLine/Secure to compare the RDBMS subject and object sensitivity label tags. The *mand_tag_to_ir* allows INFORMIX-OnLine/Secure to build the ir data structure from a subject or object tag. The *mand_ir_relationship* routine can then be used to compare the sensitivity labels of a RDBMS subject and object in internal representation form. All four functions require the **allowmacaccess** HP-UX BLS privilege.

Privilege	Program Name	Function
allowmacaccess	SQL Engine†, RSAM	The RDBMS Kernel is forked at the sensitivity label of the user initiating the session. Global shared memory and the disk are at *Datahi + IX_DATA* which are incomparable to the user's sensitivity label.
	transient processes	In order to fork the associated support process, the transient process must change sensitivity labels from **Datahi + IX_DBSA** (or **Datalo + IX_DBSA**) to **Datahi + IX_DATA**.
	datextract	Checks if the user is the DBSSO; sets the sensitivity label of the audit ouput file to **Datahi**; and extracts audit records from the OS audit trail using *audit_read.*
allowdacaccess	SQL Engine†, RSAM	The RDBMS Kernel is forked with the group equal to the group of the initiating user. Global shared memory and the disk have the group **ix_data**. The user can never be in the group **ix_data**.
	transient processes	In order to fork the associated support process, the transient process must change its group from **ix_dbsa** to **ix_data**.
	datextract	Checks if the user is the DBSSO; sets the sensitivity label of the audit ouput file to **Datahi**; and extracts audit records from the OS audit trail using *audit_read. datextract* must call *chslabel* and *getslabel* which require this privilege.
setprocident	transient processes	This privilege is required to change the group of the transient process as previously described.
writeaudit	SQL Engine†, RSAM, transient processes	All of these processes write to the OS audit log.
owner	*datextract*	This audit analysis tool must set the owner of the extracted audit file to the AAO.
multileveldir	*datextract*	Using the *ismultdir* system call, *datextract* checks a multilevel directory to verify that the output of in the directory.

† In the EP configuration, the SQL Engine and RSAM are part of the same process. In the EA configuration, only RSAM is part of the TCB and they communicate via the protocol described in Section 4.5.

Table 3.2. Use of Privileges in HP-UX BLS

The routines *mand_er_to_ir* and *mand_ir_to_er* are used by INFORMIX-OnLine/Secure to convert sensitivity labels in internal representation form to externel representation form and vice versa. The *mand_alloc_ir* and *mand_free_ir* functions are provided in order to supply the appropriate data structures to these routines.

HP-UX BLS provides a number of global identifiers to increase the efficiencey of sensitivity label manipulations and comparisons. These are: **mand_syslo**, **mand_syshi**, **mand_max_class** and **mand_max_cat**. These global identifiers are initialized whenever any of the *mand* library routines are invoked. Alternaltively, the routine *mand_init* is used by INFORMIX-OnLine/Secure to force initialization of the global identifiers. The *mand_end* routine is used when no more searches need to be done of the sensitivity label databases and the space can be freed.

3.4.2 Audit Libraries

The *authaudit* library provides an interface to the audit subsystem for protected subsystems. Since INFORMIX-OnLine/Secure has the **writeaudit** privilege, it can write directly to the audit special device, **/dev/auditw**. INFORMIX-OnLine/Secure uses the *audit_subsystem* to record an audit record. See Section 5.2.4 for a detailed discussion of auditing.

The *audit* library is provided by HP-UX BLS to allow INFORMIX-OnLine/Secure to open and access existing audit session data. Use of these library routines requires the effective user ID or group ID to be *audit* and the process sensitivity label to be **mand_syshi** unless the process has the **allowmacaccess** privilege, which is the case for *datextract* (see Section 5.3.4). INFORMIX-OnLine/Secure uses the *audit_open* routine to open an audit session that has been previously recorded on the system. Once the session is open, the *audit_read* routine can be used to retrieve audit records from the audit session data files. Before another audit session can be opened, the current session must be closed via the *audit_close* routine.

The *datextract* audit analysis tool also uses the **owner** and **makemultdir** privileges to process audit data.

3.4.3 Other HP-UX BLS Specific System Calls

The *chslabel* system call is used by INFORMIX-OnLine/Secure to change the sensitivity label of a file. In order to perform this function, the process must have mandatory and discretionary write access to the file. This system call is used by the *datextract* audit analysis tool and RSAM. These processes execute with the **allowmacaccess** and **allowdacaccess** privileges. The *getslabel* system call is used by RSAM or the SQL Engine to retrieve the current process's sensitivity label.

The *setclrnce* system call is used to set the clearance of processes. The clearance must be set prior to a *chslabel* or the *chslabel* will fail. The new clearance must be dominated by the old clearance and must dominate the sensitivity label of the calling process. This system call requires **allowmacaccess** privilege.

The *ismultdir* system call is used by INFORMIX-OnLine/Secure to check if a directory is a multilevel directory.

The routines *set_auth_parameters* and *check_auth_parameters* are used to set the appropriate authorizations for using the audit mechanism. No privilege is required for these functions which initialize all authorizations to NULL.

This page intentionally left blank

Chapter 4

Informix Database Managment System Architecture

This chapter describes the architectural components of INFORMIX-OnLine/Secure. First it provides an overview of all the Relational Database Management System (RDBMS) architectural components. After the overview, disk structures and management is described. Following that discussion is a detailed description of the four types of kernel processes: Relational Storage Access Method (RSAM) processes, support processes, transient processes, and daemon processes.

4.1 Architectural Overview

INFORMIX-OnLine/Secure supports three different security configurations. The first is the EA (Enhanced Assurance); it provides the highest degree of assurance by isolating the security relevant portion of the product. An EP (Enhanced Performance) configuration may also be installed, which adds the SQL Engine to the TCB for performance enhancement. Lastly, there is the C2 configuration. The C2 configuration is a subset of the EP configuration that does not provide MAC. Each configuration contains the same architectural components. These components are: the secure administrator front end, the administrator front end, the user front end, the SQL engine, relational storage access method, support processes, daemon processes, transient processes, and global shared memory. The next section discusses each of the architectural components.

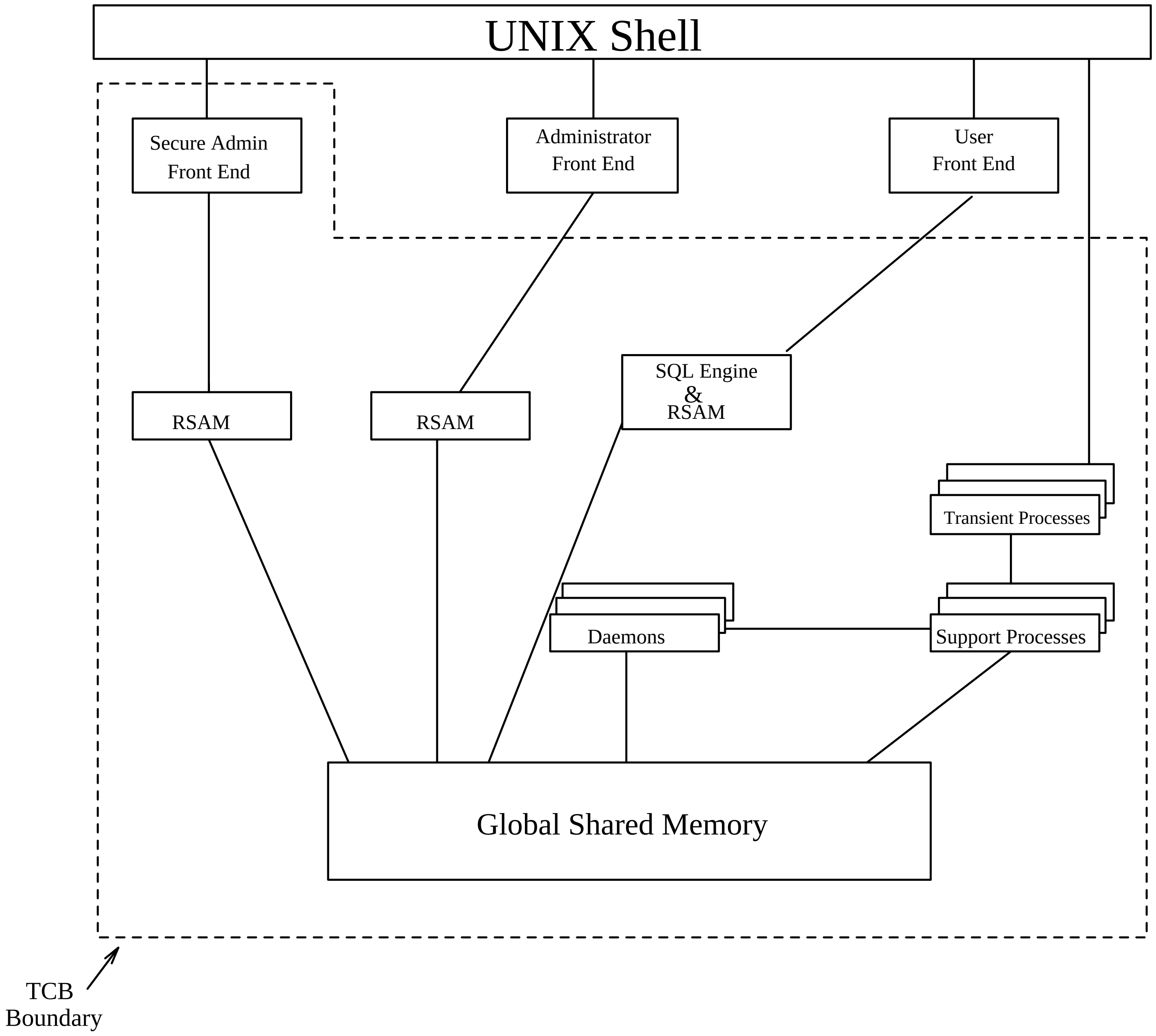
INFORMIX-OnLine/Secure runs as a trusted process with respect to a trusted OS. The following diagrams show the architectural components of INFORMIX-OnLine/Secure in relation to each other for each configuration. Next is a discussion of the components that comprise the trusted database architecture.

4.1.1 Secure Administrator Front-End

The *Secure Administrator Front End (SAFE)* is a trusted process that enables secure interaction with the RDBMS. The Database System Security Officer (DBSSO) uses the SAFE to manage sensitivity labels, access permissions, and audit events. It allows the DBSSO to determine and change sensitivity labels on RDBMS objects, and to determine and change discretionary permissions associated with RDBMS objects. The SAFE interfaces with the RSAM, and with the OS. The SAFE relies on the OS for IPC services to communicate with the RSAM processes, and for security services, such as user ID, process ID, and sensitivity label information. For more details see page 88, “Database System Security Officer and Secure Administrator Front End”.

4.1.2 Administrator Front-End and User Front-End

The *Administrator Front-End* is an untrusted interface that the DBSA may use. The DBSA may use this interface to launch transient processes. The *User Front End (UFE)* is the untrusted interface through which



.

Figure 4.1. EP and C2 Configurations

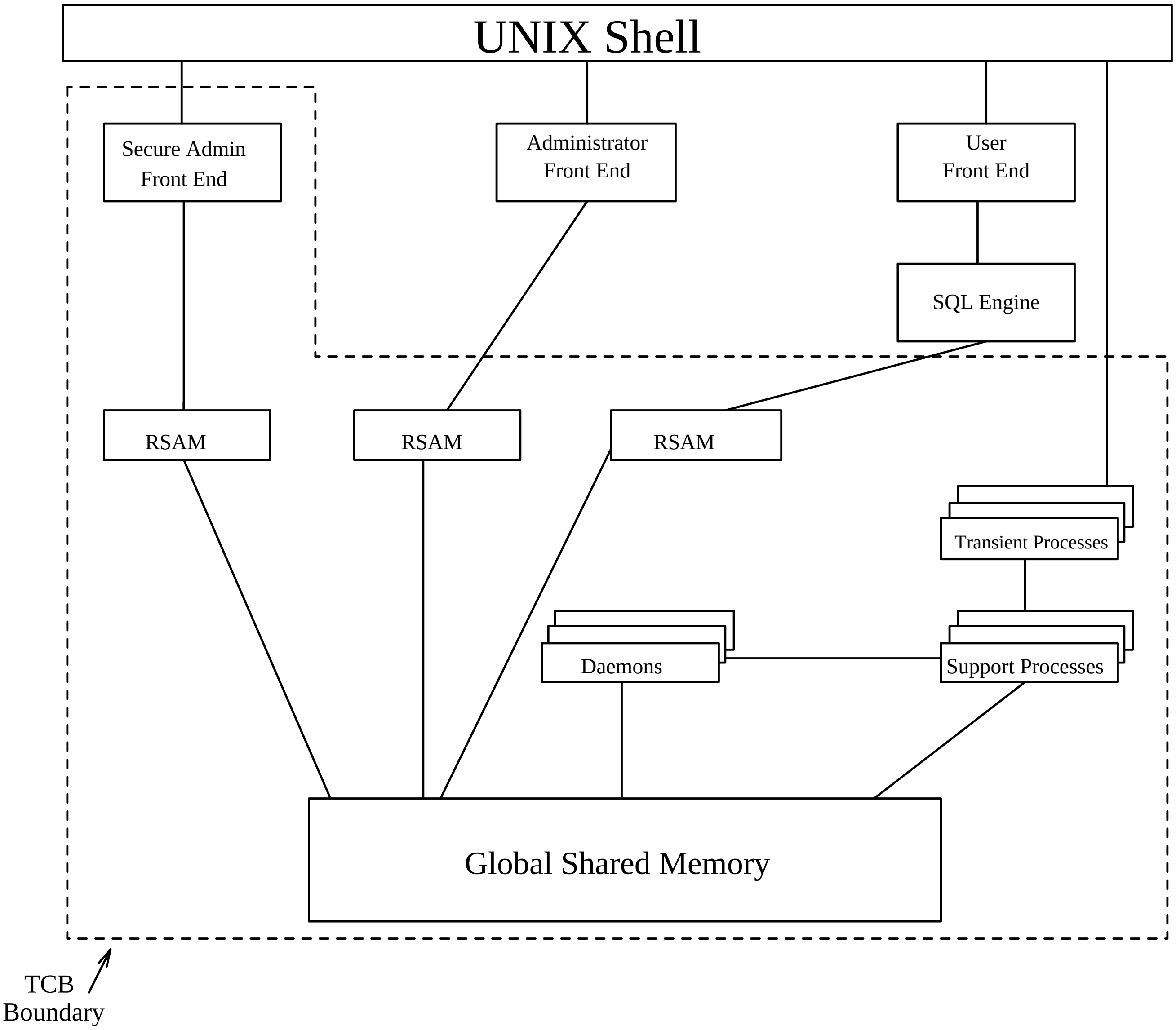


Figure 4.2. EA Configuration

ordinary users of the system interact with the database. The UFE is either menu driven or user written. The User Front-End interfaces with the SQL Engine. Users interact with the database through the SQL Engine by making SQL queries. The UFE and AFE are in all configuration; the UFE, AFE, and the SQL Engine are in the EA configuration.

4.1.3 SQL Engine

The SQL Engine plays the role of an intermediary between the User Front End (UFE) and the RSAM process, which is part of the RDBMS Kernel. The SQL Engine interprets SQL commands from the UFE, and translates the commands into data access requests sent to RSAM, the process that actually retrieves the data to satisfy the SQL command. In the B1/EP and C2 configurations, the SQL Engine and RSAM are a single process. In the B1/EA configuration, the SQL Engine and RSAM each run in their own process and have their own address space.

The SQL Engine is responsible for providing the following services:

- communicate with the UFE to receive input SQL commands
- parse the input SQL commands and perform necessary syntactic, and semantic checks
- manage SQL data definition commands that set up the database schema
- manage SQL data manipulation commands that access data stored in databases
- manage SQL commands for transaction processing
- optimize user queries to select an access path that is estimated to be the cheapest in terms of performance
- communicate with RSAM to initiate the actual data manipulation on disk
- return the query results (if any) to the UFE

The SQL Engine has three external interfaces: SQL/RSAM (EK), SQL/UFE (UE), and SQL/OS. The EK interface is between the SQL Engine and RSAM (see the discussion on page 59, “Session Startup” for more details). Data access requests are issued via this interface. These requests include data definition commands (such as creating database or altering a table schema), data manipulation commands (such as inserting or updating data), and transaction management commands (such as committing or rolling back a transaction). The SQL Engine interprets each SQL command received from the UFE, translates it into one or more low level data access requests, and forwards the request to RSAM. The ultimate task of data access is carried out by RSAM. The UE interface is between the SQL Engine and the User Front End (UFE). The UFE issues DBMS requests in the form of SQL commands via the UE interface. These commands include data definition commands, data manipulation commands, and transaction management commands as described above. The SQL Engine interprets each SQL command received from the UFE, translates it into one or more low level data access requests, and forwards it to RSAM. The SQL Engine also interfaces with the OS, which is a trusted Unix, via the Unix system call mechanism. The OS schedules and executes the SQL Engine code, manages the resources required by the SQL Engine, and provides IPC services so that the SQL Engine may communicate with RSAM and the UFE.

The SQL Engine executes in only one state: on-line. The single on-line state has three modes: start-up, operational, and shut-down. Each mode has several functions that can be performed while the SQL Engine is in that mode. Each mode, and the associated functions, will be discussed below.

The SQL Engine is not responsible for performing any access control decisions that are not validated by the RSAM. In the EP and C2 configurations, the SQL Engine is part of the TCB. The RDBMS does not perform any identification and authentication, and auditing is done by the RSAM. The SQL Engine is simply a parser or a type of complier. In the B1/EA the SQL Engine does perform DAC checks to try to optimize queries by denying a user access if a DAC check fails and by not invoking the RSAM. However, if the DAC check passes, RSAM validates this claim by performing the DAC check again. There are no object reuse issues for the SQL Engine because all data retrivals are performed by the RSAM. In the B1/EA case, the SQL Engine does not perform any unvalidated security relevant functions and must only be trusted to work properly.

4.1.3.1 Start-up Mode

The start-up mode of the SQL Engine is entered as the UFE starts an instance of the SQL Engine process. This mode has three functions: initialize of the SQL Engine data structures, fork RSAM (B1/EA only), and signal the UFE when an error is encountered. There are two reasons that an RSAM process cannot be established: the OS does not have sufficient resources or OnLine/Secure is not in the on-line or quiescent state. If an RSAM process cannot be started, the OS returns an error message and the SQL Engine error exit function returns an error message to the UFE indicating the cause of the failure.

4.1.3.2 Operational Mode

The operational mode of the SQL Engine supports users in performing their daily duty of database access. The following functions are part of this mode: UE interface maintenance, parse SQL commands, dispatch, optimize query, data definition, data manipulation, transaction management, concurrency management, maintain in-core dictionary, and EK interface maintenance. Each of these functions will be discussed in this section. It is important to note that the SQL Engine is managing the SQL commands received from the UFE and translating these commands into one or more low level data access requests. The actual data access functions are performed by RSAM, not the SQL Engine.

UE Interface Maintenance This function essentially sleeps and waits for user input from the UFE. When a message from the UFE arrives through the communication pipe (established between the UFE and the SQL Engine), this function is invoked and passes the message onto the parsing function for processing. This function is also invoked when status information must be sent back to the UFE. This function prepares a message that contains the status/results and sends the message back across the pipe.

Parse SQL Command This function is invoked to check the syntax and semantics of the SQL command, and to build a parse tree. The SQL command is first checked to make sure it is syntactically compatible with INFORMIX-SQL grammar which conforms to level I of the ANSI database Language SQL standard. The grammar is written in YACC. Subsequently, the input statement is passed to the check semantics function. The semantic checks are implemented by the YACC grammar actions. They are performed during the parse time over the input statement, rather than during a second pass over a parse tree generated in the first pass. Because of this, it is sometimes necessary to "look ahead". The FROM clause within a SELECT statement illustrates this point.

In order to perform the semantic checks, the database schema stored in the data dictionary must be available. The SQL Engine invokes the in-core dictionary maintenance function to handle this. After the input statement is successfully parsed for semantic consistence, it is passed to the build parse tree function. This function constructs an internal data structure to represent each input SQL statement. This data structure is generically called a parse tree, although it is sometimes also referred to as a control block. Once a parse tree is built, the subsequent processing of the statement takes place on this internal representation rather than the original statement text. Everything about a statement is stored in the parse tree that represents it.

Parse trees do not persist. They are built in memory heap, which is allocated in large blocks to avoid fragmentation. Since a parse tree is generally needed throughout the entire execution of a statement, memory allocated through the heap is not freed until the processing is completed. After a parse tree is successfully built, the dispatch function is invoked.

Dispatch Function The dispatch function receives a parsed SQL command that is one of four kinds: data definition, data manipulation that requires optimization, data manipulation that does not require optimzation, or transaction management. The dispatcher forwards the request to the SQL Engine function depending on this command type. Note that the SQL Engine functions only processes the command itself but does not perform any data access functions; data access requests are passed to RSAM for data manipulation.

Optimize Query This function has two purposes: to select an optimal access path and to build a query plan. Selecting an optimal access path means to choose an access path that is estimated to be the cheapest in terms of performance. This is particularly important when multiple tables are involved in a query because the order in which the tables are accessed can make a significant difference in performance. Once the access path has been determined, a query plan is constructed that represents the chosen access path. The query plan is implemented as an internal data structure that tells how to process the query. Once the query plan is successfully constructed, it is passed to the data definition or data manipulation function.

Data Definition and Data Manipulation Data definition operations include the following: create database, drop database, create table, drop table, alter table, rename table, rename column, create index, drop index, alter index, create view, drop view, create synonym, drop synonym, grant privilege, and revoke privilege. Data manipulation operations that require optimization are: select row, delete row, and update row. Those that do not require optimization include: insert row, load table unload table, and update statistics. If the operation was optimized, a query plan will be carried out. Otherwise a parse tree will be used. The role of the SQL Engine at this point is to manage the data access requests that are passed to RSAM for processing. The SQL Engine does not carry out the data definition or data manipulation commands.

DAC and Auditing In the B1/EA configuration, the SQL Engine does perform DAC checks in an attempt to optimize queries. The SQL Engine performs DAC checks on user queries and if the check fails it denies the user access and generates an audit record. In the event the DAC check passes, the RSAM validates this claim by performing the DAC check again before allowing the user access to data.

Transaction and Concurrency Management The transaction management function handles SQL commands for begin transaction, commit transaction, rollback transaction and to set the logging mode. Concurrency commands include: lock/unlock table, set isolation level, and set lock mode. If the operation was optimized, a query plan will be carried out. Otherwise a parse tree will be used. This function of the SQL Engine manages transaction and concurrency management commands. The actual processing is preformed by RSAM.

In-Core Dictionary The in-core dictionary is in the SQL Engine's private address space. Therefore, different user sessions cannot share information in the in-core dictionary even though the session may access the same tables. The dictionary is a linked list of table descriptors for user tables (both temporary and permanent). The data in the table descriptor is obtained from the system catalogs. When a table is referenced for the first time, a table descriptor is built for the table and placed at the head of the linked list. There is only one entry for a user table in the in-core dictionary irrespective of the number of accesses to the table.

Each time a table is accessed, the in-core dictionary is accessed. During each such access, a check is made to determine if the information in the in-core dictionary is stale as compared with the corresponding data on disk. The in-core is stale if the version number of the table in the in-core is less than the version number in the systables catalog on disk. If the in-core is stale, a new table descriptor is built and the stale version is dropped. The entire table descriptor has to be updated at once.

EK Interface Maintenance this function is responsible for managing the EK interface between the SQL Engine and RSAM. The request is formulated by the SQL Engine and passed to RSAM based on the configuration. In the B1/EP and C2 configurations, subroutine calls are used. In the B1/EA configuration a message passing protocol is used.

4.1.3.3 Terminate Processing (Shut-Down) Mode

This function simply terminates the child RSAM process and removes the shared memory segment used for communication between the SQL Engine and RSAM. This is true only for the B1/EA configuration. The SQL Engine then exits. In the B1/EP and C2 configurations, the combined RSAM/SQL Engine process is destroyed. There is no shared memory to remove and no separate RSAM to kill.

4.1.3.4 Other SQL Engine Security-Relevant Features

Cursors A cursor is an identifier associated with a set of rows. Conceptually, it is a pointer to the current row. There are three types of cursors: sequential, scroll, and hold. Users are not permitted to share cursors.

A sequential cursor is used to fetch only the next row in the sequence from an active set. The sequential cursor can only read through the active set once each time it is opened. When using a sequential cursor, it returns the current row and locates the next row in the active set. The scroll cursor is used to fetch rows from the active set in any order. Unlike the sequential cursor, rows may be fetched in any order and the cursor remains open (i.e., it points to the current row). A hold cursor remains open past the end of a transaction; it allows uninterrupted access to a set of rows across multiple transactions.

Isolation Levels The isolation level selected by the user determines the level of data stability required. Four levels of data isolation are supported: *dirty read*, *committed read*, *cursor stability*, and *repeatable read*. Each level of isolation provides an increased degree of isolation from other users who might simultaneously be accessing the same data in the RDBMS.

The dirty read level of isolation provides the lowest level of concurrency overhead by not providing a guarantee on the integrity of the data. Data accessed at this isolation level is returned to the requestor regardless of any other access or locking currently in force. As a result, no interrogation or modification of any existing locks is required. This type of isolation is particularly useful when accessing largely static tables since the inherent risk of data corruption is minimized. In addition, the non-intrusive nature of dirty read allows data to be read at a lower sensitivity label than that of the current session without the risk of a covert channel. If a dirty read is done on a row during a transaction that is later rolled back, the user has no way of knowing the row was not committed. This type of row is called a phantom row.

A committed read is the default isolation level. Data accessed under this level of isolation is only returned to the requesting process if there are no exclusive locks currently in force on the requested data. A committed read ensures the user that any row returned to the requestor is not affected by any currently pending transactions and ensures that no phantom rows are returned. However, since only the ability to create a shared lock is checked and a lock is not actually created, there is no guarantee that the row is not subsequently modified by another process.

The cursor stability isolation level ensures that a row's data remains stable as long as it is the current row in a cursor-based transaction. Once the current row moves on to another row or the transaction is completed the shared lock is released. This lock is invisible if the data being accessed is strictly dominated by the sensitivity label of the current session.

The highest level of isolation is the repeatable read. At this isolation level, data stability is ensured throughout an entire transaction. This is achieved by acquiring exclusive locks on the data. The locks are invisible if the data being accessed is strictly dominated by the sensitivity label of the current session. The invisible lock does not prevent a process at a lower sensitivity label from querying and modifying the data held by the higher sensitivity label process. Instead, the originating process (the higher sensitivity label process) is informed that the data has been changed and that the process may want to roll back to preserve the semantics of the repeatable read isolation level.

4.1.3.5 Relational Storage Access Method

The *RSAM process* acts as a server to a Secure Front-End process, an Administrative Front-End process, or an instance of the SQL Engine running on behalf of a user process. There is one instance of an RSAM process per user session. Each RSAM process is multi-level since it needs to communicate with the front-end processes and global shared memory at various sensitivity labels. See page 39, “Relational Storage Access Method (RSAM) Processes” for an explanation of why RSAM is considered a multi-level process.

RSAM is the heart of the RDBMS. It performs many types of functions including fault tolerance services, database abstraction services, transaction management services, disk management, cache management, and concurrency management. The design and functionality of RSAM is described further in page 39, “Relational Storage Access Method (RSAM) Processes”.

4.1.3.6 Support Processes

Support processes are small specialized programs that perform infrequent non-periodic tasks. They are single threaded UNIX processes. Support processes are invoked by the Database System Administrator (DBSA) through transient processes. The invoking transient process will spawn a support process only when the transient process has the correct sensitivity label for doing so. The support processes are trusted because they have access to multi-level database data on disks, archive tapes, and initialization data during start-up.

See page 60, "Support and Transient Processes" for more details.

4.1.3.7 Daemon Processes

Daemon processes are specialized programs that conduct repetitive tasks to service all RSAM processes. They are single threaded and are implemented in separate UNIX processes. Daemons are started when INFORMIX-OnLine/Secure is initialized; they remain in execution until they are explicitly terminated by the DBSA or there is a system shutdown. The number of daemons is not dependent on the number of user sessions; instead, the number is configured by the DBSA during system configuration. For more details see page 67, "Daemons".

4.1.3.8 Transient Processes

Transient processes are small programs which have the single function of launching support and daemon processes. The transient processes are MAC-exempt which allow them to change to the appropriate sensitivity labels before spawning the daemon or support processes; this is in order for the daemon and support processes to run at a single-level. Transient processes only interact with the UNIX shell and the TCB to release an instance of a daemon or support process. Transient processes are described in detail on page 60, "Support and Transient Processes".

4.1.3.9 Global Shared Memory

Global shared memory is a large segment of main memory that is shared among multiple kernel processes (RSAM processes, support processes, daemon processes, and transient processes). Utilizing a shared memory scheme reduces disk accesses and pages are buffered on a system-wide basis rather than a per-process basis. Global shared memory is allocated statically during INFORMIX-OnLine/Secure initialization, and its size does not grow or shrink dynamically. For more information on global shared memory, see page 40, "Global Shared Memory".

4.2 System States

In order to support processing, INFORMIX-OnLine/Secure can exist in seven states. Each of these states supports a specific type of interaction with a specific set of users. The seven states are: off-line, start-up, quiescent, on-line, shutdown, restore, and abort. The states are shown in Figure 4.3.

The off-line state is the state in which INFORMIX-OnLine/Secure is in before it starts up. It is a single user state with only the DBSA's actions supported. The only function in this state is the ability to change to another state, namely the start-up state.

In the start-up state, INFORMIX-OnLine/Secure transitions from the off-line state into either the restore or quiescent state. In the start-up state, INFORMIX-OnLine/Secure takes direction from the DBSA and configuration files. There are seven functions associated with this state: restore from archive, read system parameters, initialize cache, abnormal termination check, fast recovery, change state and audit.

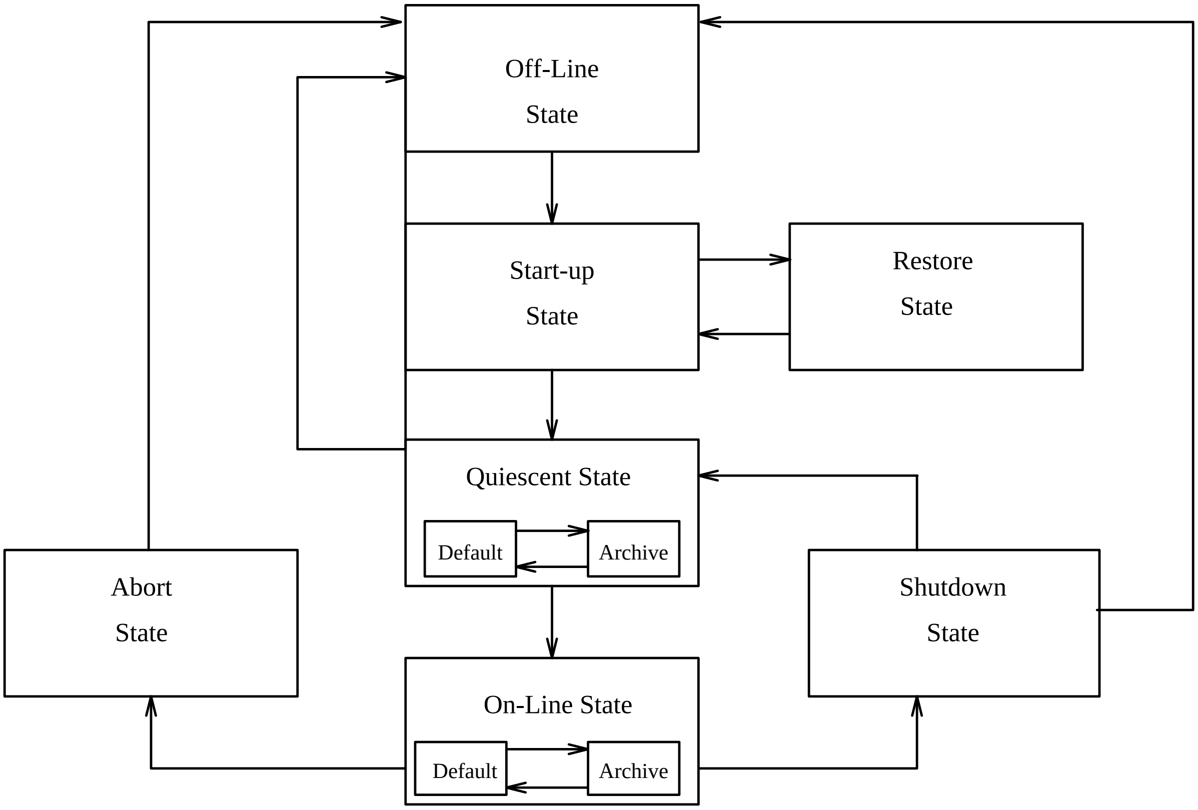


Figure 4.3. State Relationships

The restore state is a transitory state that may only be reached from the start-up state. It has three functions, restore system from archive, audit, and stop processing.

In the quiescent state, only interactions with the DBSA or DBSSO are allowed. This state allows the DBSA or DBSSO to change system parameters or perform maintenance that must be done while users are not accessing the system. The functions associated with this state are archive system, set default logging mode, view audit events, modify audit events, view object sensitivity label, modify object sensitivity label, view Discretionay Access Control (DAC) object permissions, change DAC object permissions, change state, and audit.

The on-line state is where INFORMIX-OnLine/Secure is fully functional and interactions are allowed with users of the system, the DBSA, and the DBSSO. This state supports all activities by users requiring access to the database. INFORMIX-OnLine/Secure must be in the on-line state to support the RDBMS functionality. To accomplish the support of user activities, numerous functions such as data management, auditing, and object reuse are supported. The DBSA can transition the RDBMS from this state into either the abort or shutdown states.

To transition INFORMIX-OnLine/Secure from the on-line state into one of the non-interactive states, the shut-down state is used by the DBSA. The two functions allowed in this state are stop processing, and change state. The two states that INFORMIX-OnLine/Secure may transition into are the quiescent state and the off-line state. If processing is stopped all user processes are disconnected and all transactions are rolled back to the last committed state. After this, INFORMIX-OnLine/Secure moves into the quiescent state.

The abort state is entered when it is necessary to terminate user interactions with INFORMIX-OnLine/Secure immediately. This state has three functions including audit, terminate user, and change state. To terminate user actions, INFORMIX-OnLine/Secure kills all user processes and rolls transactions back to the last committed state.

4.3 Disk Management and Structures

INFORMIX-OnLine/Secure uses its own mechanisms for managing storage space. Any number of databases, tables, and rows at different sensitivity labels can be contained in the same OS structure. INFORMIX-OnLine/Secure has a number of structures and mechanisms for maintaining and separating Database Management System (DBMS) data within the OS structures. This section describes the disk structures and disk management tasks that INFORMIX-OnLine/Secure performs.

Space from disks and files is assigned to INFORMIX-OnLine/Secure in units called *chunks*. A collection of chunks is called a *dbspace* which contains databases and tables. Chunks are broken down into smaller units called *pages*. A page is the basic unit of I/O for INFORMIX-OnLine/Secure. The complete set of pages allocated to a table is a set of *tblspaces*. When a table needs to expand, a set of pages called an *extent* are allocated for the table and the tblspace is increased by the extent size. Special dbspaces called *BLOBspaces* can be used to hold Binary Large Objects (BLOBs). Figure 4.4 shows the relationships between all the disk components mentioned above.

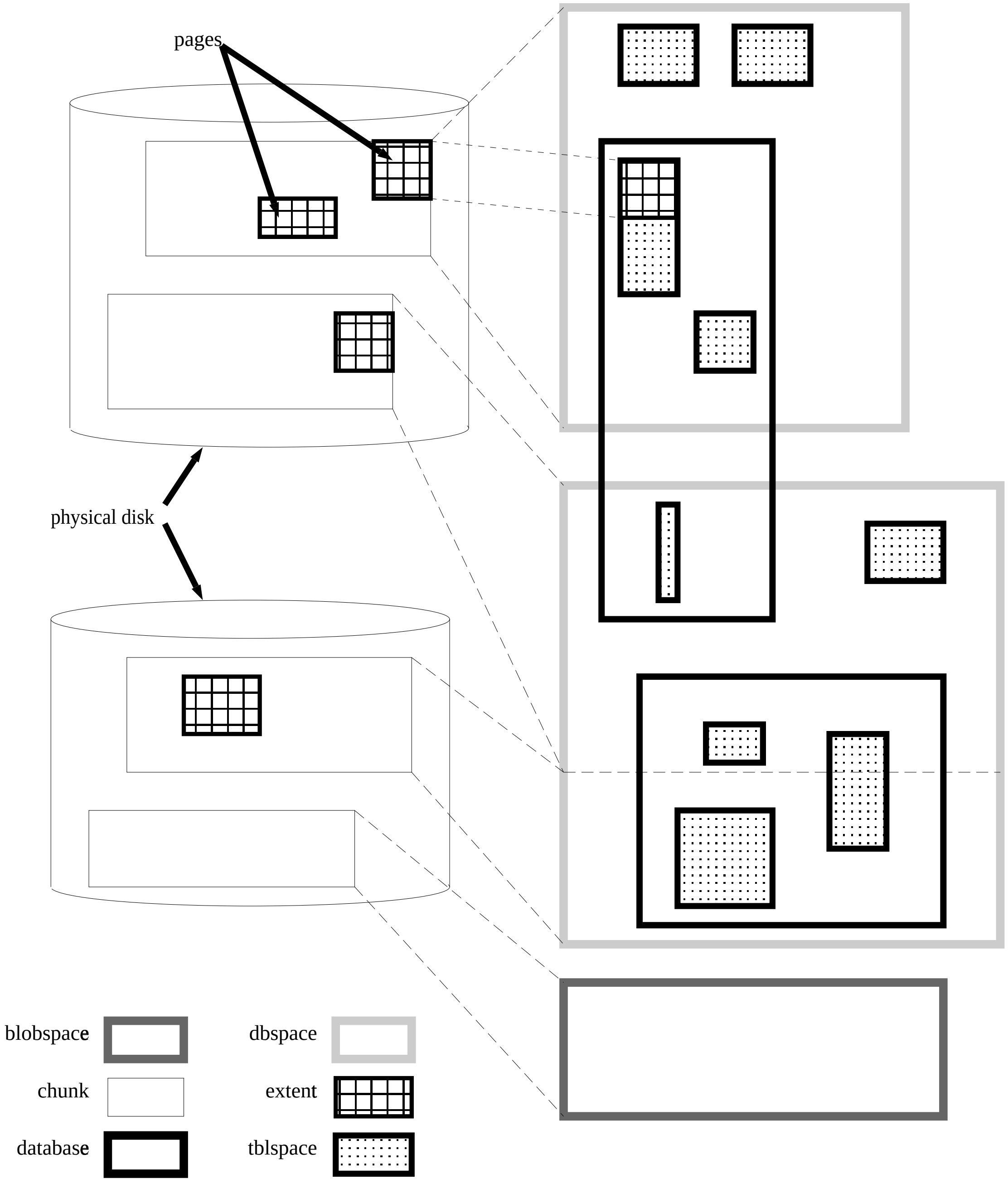


Figure 4.4. Relationships between physical and logical disk components.

4.3.1 Chunk Management

A chunk is a unit of space that is provided to INFORMIX-OnLine/Secure by the OS. The OS needs to provide an abstraction that allows random access, can be identified by a pathname, and can be referenced by that pathname or a descriptor. A chunk can be a disk partition or a UNIX file.

The use of chunks is closely tied to the use of dbspaces. This section describes how chunks are defined to INFORMIX-OnLine/Secure. Detail on the layout of information in chunks is found in the dbspace description.

Chunks are defined by a pathname (the name of the device or file), a starting offset (distance in KB from beginning of device), and a size (amount of space in KB). Only INFORMIX-OnLine/Secure knows how to manage and interpret the information in a chunk. Initialization of a chunk by INFORMIX-OnLine/Secure involves initializing reserved space used by INFORMIX-OnLine/Secure and marking the rest of the available chunk space as free. The reserved space includes information to keep track of the free space on the chunk along with configuration data.

INFORMIX-OnLine/Secure issues UNIX system calls to open and close the chunk and is subject to the underlying OS's security policy for access to the chunk partition or file. Access to chunks is controlled using the OS Mandatory Access Control (MAC) and DAC controls so that only INFORMIX-OnLine/Secure can access information stored in a chunk. See page 87, "Operating System Administrator" for details on the protection attributes used for chunks. The file descriptor returned from the UNIX *open* system call is then used for access to the chunk itself.

4.3.2 Dbspace Management

Dbspaces are logical entities that represent a collection of one or more chunks for storing databases and tables. Each dbspace must have at least one chunk assigned to it which is known as the *root chunk*. The chunks assigned to a dbspace define the physical space available to databases and tables. Chunks can be added to a dbspace if necessary. There are two types of dbspaces: regular dbspaces and BLOBspaces. This section discusses regular dbspaces and page 37, "BLOBspace Management" discusses mechanisms used for BLOB storage management.

There is a special dbspace called the *root dbspace* that contains system initialization information. The pathname of the root chunk of the root dbspace is stored in a configuration file so that INFORMIX-OnLine/Secure can find that chunk and retrieve the initialization data.

Some of the space allocated to a dbspace is used by INFORMIX-OnLine/Secure to keep track of data in that dbspace. System information, space management information and the *tblspace-tblspace* are the three types of information kept in reserved space.

The root chunk of the root dbspace also contains a database tblspace, an audit tblspace, physical log space, and logical log space. The database tblspace is used for tracking all the databases created in INFORMIX-OnLine/Secure. The audit tblspace contains audit masks and their associated names (see page 85, "Auditing").

The logical log contains a record of logical operations performed during INFORMIX-OnLine/Secure processing. If a database is created with the option to log transactions turned on, all transaction information is stored in the logical log. The physical log contains before images of pages that have been modified during processing. When the physical log before images are combined with the most-recent records stored in the logical logs, INFORMIX-OnLine/Secure can return all data to a consistent state. See page 43, "Logical Log Buffer" for more information on logical and physical log use.

4.3.3 Tblspace Management

A tblspace is a logical entity used to refer to a collection of space allocated for a table. A tblspace resides within a dbspace and is defined by extents. Extents are a physical entity representing a set of contiguous pages from a chunk. The extents assigned to a tlbspace can come from any chunk in the tblspace's dbspace.

The size of extents for a tblspace are specified at table creation. Two sizes are specified, the initial extent size and the size for additional extents. When all the extents in a tblspace are full a new extent is allocated from the tblspace's dbspace. If there is no space available in the dbspace INFORMIX-OnLine/Secure stops all processing and transitions into the quiescent state. The minimum size of an extent is 8 KB and the size must be an even multiple of the page size. A tblspace has a limit on the number of extents it can have dependent on the OS on which INFORMIX-OnLine/Secure is executing.

There is one special tblspace, the tblspace-tblspace, created in the root chunk of a dbspace as part of the dbspace creation process. Each entry in a tblspace-tblspace is one page long and describes a tblspace in the dbspace. The first entry describes the tblspace-tblspace itself. The first extent of the tblspace-tblspace starts at a fixed location in the root chunk of the dbspace so that it is readily accessible.

Tblspaces are identified using a tblspace number. The tblspace number corresponds to the logical page number of the tblspace-tblspace page that describes that tblspace. The number is composed of a dbspace number (starting from one for the root dbspace) and a sequential tblspace number within the dbspace (starting from one for the tblspace-tblspace itself).

The second tblspace described in the root dbspace tblspace-tblspace is the database-tblspace. The database-tblspace describes all the databases defined in an instance of INFORMIX-OnLine/Secure. Each row in the database-tblspace describes one database. Dbspaces other than the root dbspace do not have a database-tblspace.

The third tblspace described in the root dbspace tblspace-tblspace is the audit-tblspace. This tblspace contains information about all the audit masks defined. The information is stored as rows that contain an audit mask name and 4 32-bit words for the audit mask itself. See page 85, "Auditing" for more details about audit masks.

When a tblspace is created the initial extent is allocated and the tblspace-tblspace page is initialized. To add space to a tblspace a free extent is found in the current dbspace. When a tblspace is dropped all the tblspace's extents are added to the list of free extents in the chunk.

4.3.4 Page Management

The basic unit of I/O and disk storage that INFORMIX-OnLine/Secure uses is a page. A page is a physical entity used to store user and system data. The size of a page is determined by Informix depending on the computer that INFORMIX-OnLine/Secure is running on. The type of data stored on a particular page are homogeneous and can be row data, index information, or administrative data.

Tblspace pages are assigned sequential logical page numbers within the tblspace. The physical page address is made up of the chunk number and the page number within the chunk. The mapping from logical page number to physical page number is performed with the information about tblspace extents kept in the tblspace-tblspace.

Page types can be functionally grouped into the five categories below:

- data pages
  - tblspace row data pages
  - in-tblspace BLOB pages
  - BLOBspace BLOB pages
- index pages
- system data pages
  - chunk and dbspace reserved pages
  - tblspace-tblspace pages
- space management pages
- free pages

In-tblspace BLOB pages and BLOBspace BLOB pages are described on page 37, "BLOBspace Management".

4.3.4.1 Tblspace Data Pages

Data pages store the rows for tblspaces. Each row is identified by a rowid made up of the logical page number and the slot number of that row. The physical position of the row on the page may change due to compression of empty space on the page, but the rowid never changes for a given row. Whenever possible a row is placed in a page in its entirety. If a row cannot fit on the remaining space of a page a new page is allocated for the row. For rows that are longer than one page remainder pages are used. A forwarding pointer is used to indicate in which page and slot the remainder of a row is located. A forwarding pointer is placed before the data in the slot of the initial page for the row. When a row is updated and no longer fits on its original page the indirection indicator bit in the row's slot table entry is set. This bit tells RDBMS Kernel that the slot contains a pointer to the location of the row instead of the row itself.

Figure 4.5 gives examples of full data pages, partially full data pages, full remainder pages, and partially full remainder pages.

4.3.4.2 Index Pages

Pages containing the information that makes up an index are known as index pages. Index pages only exist in a tblspace if an index has been created for the table. Index entries are stored on index pages similar to the manner in which data is stored on data pages. Each index entry is placed in a slot on the page. Index pages are not directly accessed by users. The data stored in index pages are used by the DBMS kernel for speeding up queries into a table and are not returned to the user.

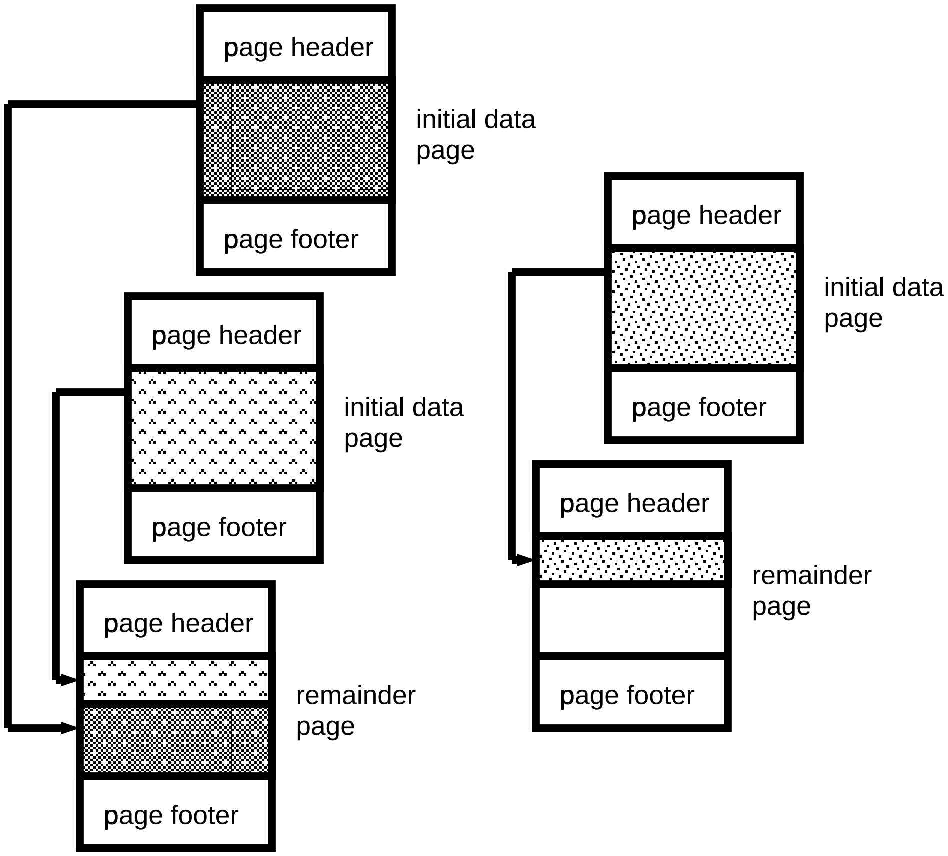


Figure 4.5. Examples of tblspace pages.

4.3.4.3 Space Management Pages

Space management pages are used to keep track of extents in a chunk, pages in a tblspace, and BLOBspace pages in a BLOBspace. The space management page indicates whether space is unused, partially full, or unavailable. The space management page used to track pages in a tblspace also indicates what kind of data is on a page.

4.3.4.4 Free Pages

Free pages are pages that have been allocated as part of an extent but have not yet been assigned any data. These pages can be used for any purpose. Pages that have been allocated and become empty revert back to being free pages.

4.3.5 BLOBspace Management

A BLOB is an unstructured, variable length sequence of bytes which can be of virtually any size. BLOBs are not stored as part of a row. Only the information required to access them is stored in the row. A *tuple BLOB structure* containing the BLOB size, the address of the first page the BLOB uses, and the time stamp of that first page is placed in the BLOB's row.

BLOBs can be stored in the same tblspace where the row to which they belong is stored or in a separate space called a BLOBspace. BLOBs are stored as sets of pages linked together. To guarantee that all of the pages are retrieved when accessing a BLOB a BLOB time stamp scheme is used. When a BLOB page is reused its BLOB time stamp is incremented. Each page containing BLOB data contains, along with its own time stamp, the time stamp the next BLOB page should have. This time stamp is compared to the actual time stamp of the next BLOB page when that page is retrieved. Different time stamps indicate that the BLOB page has been re-allocated to a new BLOB, or to a BLOB being created, and that the current BLOB retrieval should be abandoned.

When a row with a BLOB column is inserted in a table the BLOB is created first. The row in kernel memory is then updated with the tuple BLOB structure and the row is inserted into the tblspace for the table. When a row is updated, the addition of any new BLOBs is performed in the same manner as for the insertion of a row with a BLOB. The information in the old tuple is used to remove any BLOBs that were replaced. The creation of the BLOB and the creation (or update) of the row are treated as one operation to RDBMS Kernel and if either step fails the whole operation is aborted. The tblspace and BLOBspace pages used to create a BLOB are retrieved from the set of available pages and their entire contents are overwritten.

A user deletes a BLOB by deleting the BLOB's row or by updating the BLOB's entry in the row to NULL. Each of the BLOB's pages is marked as free, but data is not erased on those pages. See page 82, "Object Reuse" for more details.

The operation of reading a BLOB is also two steps. To prevent unauthorized access to a BLOB the RDBMS Kernel only allows access to the BLOBs pointed to by the last row that RDBMS Kernel retrieved for a user. RDBMS Kernel knows the user is authorized to access the BLOB since RDBMS Kernel determined the user was authorized to access the row. After confirming a user can access the requested BLOB, BLOB pages are read in. The RDBMS Kernel compares the time stamps after each read, if any time stamps do not agree, the read is aborted at that point.

There is a situation where a user may read a deleted BLOB. As described above, reading a BLOB requires two steps. User A can read a row with a BLOB and receive the tuple-BLOB structure. User B can delete the BLOB (as described above). User A can then request the BLOB giving the tuple-BLOB structure received earlier. The RDBMS Kernel will retrieve the BLOB. Since the BLOB was deleted some pages could have been reused in the creation of another BLOB. If this is the case the time stamps on the reused pages will have been changed and will not agree with the rest of the time stamps in the deleted BLOB and the retrieval is aborted.

Management of BLOBs stored in BLOBspace pages and in-tblspace pages is different. The different disk structures and use of the pages for each is described below.

4.3.5.1 In-tblspace BLOB Pages

In-tblspace BLOB pages are typically used for BLOBs when the BLOB is small. The contents of a BLOB are stored in a linked list of one or more tblspace pages. The tuple BLOB structure stored in the BLOB's row points to the first page that contains the BLOB's data. The page format for a BLOB page is similar to regular data pages.

For portions of a BLOB other than the last, a single slot in the BLOB page contains all of the BLOB data that fits on that page along with an in-tblspace BLOB page header. Final parts of in-tblspace BLOBs can share a common BLOB page. These BLOB portions are stored in separate slots of the shared page.

4.3.5.2 BLOBspace BLOB Pages

A BLOBspace is a special dbspace used to handle the I/O of large BLOBs more efficiently. BLOBspaces are defined as a collection of chunks just like dbspaces and the entry in the root chunk of the root dbspace contains a flag to indicate a BLOBspace rather than a dbspace. BLOBspace BLOBs are stored in storage units called BLOBspace BLOB pages (referred to as BLOBspace-pages in the rest of this section). BLOBspace-pages can be any multiple of a dbspace page size and all the BLOBspace-pages of a given BLOBspace will be the same size.

A BLOB in a BLOBspace is stored as a linked list of BLOBspace-pages. The first BLOBspace-page of the BLOB is found using the tuple BLOB structure. A BLOBspace-page consists of an integer number of consecutive pages in the chunk. This number is specified at BLOBspace creation time and is constant for all the BLOBspace-pages in that BLOBspace.

4.3.6 Mirroring

Chunk mirroring is the maintenance of an identical copy of a chunk in another chunk. This copy of the information is used to prevent loss of information as the result of a disk failure. Chunk mirroring automatically replicates data to a different chunk (called the mirrored chunk) as it is written to the original chunk (called the primary chunk). If either chunk has a hardware failure, INFORMIX-OnLine/Secure just continues processing with the other chunk. A hardware failure is detected by checking the return code after every write to the chunks.

After a hardware problem (or when a chunk is being mirrored for the first time) the mirror chunk is brought up to date with the primary chunk. This involves opening the mirror chunk and copying all the pages from the primary chunk to the mirror chunk. From that point on all writes to the primary chunk are also performed on the mirror chunk.

4.4 Relational Storage Access Method (RSAM) Processes

There are four types of processes within the RDBMS Kernel: Relational Storage Access Method (RSAM) processes, daemon processes, support processes, and transient processes[1]. Daemon processes are discussed on page 67, "Daemons". Support and transient processes are discussed on page 60, "Support and Transient Processes". This section discusses the RSAM processes.

A user must be in the group **ix_users** to initiate a database session. In the B1 configurations, a user process must also have a sensitivity label in the range from **Datalo** to **Datahi** to initiate a database session. In the C2 and EP configurations, when a user initiates the SQL Engine, RSAM is part of the same process as SQL and is part of the TCB. So, a single process with a sensitivity label equal to that of the user is created. In the EA configuration, when the user initiates the SQL Engine, the Engine forks a separate RSAM process. Both processes have a sensitivity label equal to that of the user process initiating the database session. In this configuration, RSAM is a process that is part of the TCB while the instance of the SQL Engine is a user process and not part of the TCB. In this configuration, RSAM and the SQL Engine are distinct processes communicating via the protocol described on page 59, "Session Startup".

The SQL/RSAM process in the C2 and EP configurations and the RSAM process in the EA configuration, has privilege that allows it to bypass MAC and access the raw device, global shared memory, and other RDBMS data structures such as the **tbconfig** file, which have a sensitivity label of **Datahi** + **IX_DATA**. Thus, it is called a "multilevel process" because it has privilege to access data at any sensitivity label as needed[2].

In all configurations, when INFORMIX-OnLine/Secure is in either the quiescent or on-line state, the RSAM process acts as a server to the front-end processes and dispatches work among the possible functions.

The DBSSO uses the Secure Administrator Front-End (SAFE) interface; the DBSA uses the Administrative Front-End (AFE) interface; and, in general, regular users use the SQL Engine[3]. When an RSAM process is created, the user's group and the category portion of the user's sensitivity label are used by RSAM to determine the type of user on whose behalf it is being created. An RSAM local variable is set to a one if it is a DBSA, a two if it is a DBSSO, and a zero if it is a regular user. RSAM subsequently uses this local variable to determine if requested DBSSO and DBSA operations are permitted. There is little overlap in the set of functions allowed for each type of user[4].

[1] Throughout the rest of this report, when all four processes are being refered to the term "RDBMS Kernel" is used; when a specific process within the the RDBMS kernel is being discussed, such as RSAM, daemon process, support or transient process, the particular process name is used.

[2] Henceforth, the term RSAM will refer to both the SQL/RSAM process in the C2 and EP, and the RSAM process in the EA.

[3] A user may interface with RSAM through a third-party tool, or their own tool, provided the tool is compatible with the UFE/Engine interface.

[4] There are actually five functions that are common to all three interfaces, but they are considered security irrelevant. They are: begin transaction (*isbegin*), commit transaction (*iscommit*), rollback transaction (*isrollback*), convert between BLOBspace name and number (*isbspace*), convert between dbspace name and number (*isdbspace*).

Global Shared Memory Header
User Process Table
Lock Table
Dbspace Table
Chunk Table
Mirror Chunk Table
Tblspace Table
Buffer Table
Buffers
Physical Log Buffer
Logical Log Buffer
Flush Control Structure

Figure 4.6. Global shared memory organization

4.4.1 Global Shared Memory

In INFORMIX-OnLine/Secure a large segment of main memory is shared among multiple RDBMS Kernel processes and is accessible to every active RDBMS Kernel process. Using a shared memory scheme reduces disk I/O since data and index pages are buffered on a system-wide basis rather than a per-process basis. In addition, process synchronization is less expensive since it can be implemented through global shared memory[5], rather than an interprocess communication mechanism.

Global shared memory is allocated statically during intialization. In order to access global shared memory, each RDBMS Kernel process must first attach to it. A *key* is calculated by the RDBMS Kernel process from the server number found in the **tbconfig** file. The tbconfig file has a sensitivity label of **Datahi + IX_DATA** and is owned by group **ix_data**. Since no users have the category **IX_DATA** and the group **ix_data** has no members other than root, no one other than the RDBMS kernel processes (see above) can attach to the global shared memory. The key found in the **tbconfig** file is supplied to the OS *shmat* system call in order to attach to the global shared memory. The size of global shared memory does not grow or shrink dynamically, i.e., the size of the tables that reside in the global shared memory are fixed and determined by the Database System Administrator (DBSA) during INFORMIX-OnLine/Secure initialization. Entries in global shared memory tables and data structures are marked as free or invalid by a flag.

The organization of global shared memory is shown in Figure 4.6. The global shared memory header stores housekeeping information, primarily about the global shared memory itself, which includes:

- A *latch* used to prevent multiple RDBMS Kernel processes from modifying the global shared memory header simultaneously,
- The total size of the global shared memory,
- The process ID of the initialization process that created the global shared memory as well as the associated user ID,

[5] Global shared memory data structure names are capitalized to distinguish them from RSAM local memory data structures.

- The sizes of the tables stored in global shared memory,
- Pointers to the tables stored in global shared memory,
- System-wide profiling information (number of disk reads/writes, cache hits/misses, B-tree operations, etc.),
- Audit mask structure (default, compulsory, DBSA).

4.4.1.1 The User Process Table

For each active RDBMS Kernel process, there is an entry in the global shared memory User Process Table which indicates the user on whose behalf the process was initiated. Every time an RDBMS Kernel process is started, a free entry in the global shared memory User Process Table is allocated for it. The entry is released for reuse when the process exits. If the maximum number of active processes is reached, a new RDBMS Kernel process cannot be started until one of the active processes exits and a free entry becomes available. As described above, each RDBMS Kernel process may be on one of three possible lists, waiting for a lock, a latch, or a buffer.

4.4.1.2 The Lock Table

Concurrently executing RDBMS Kernel processes synchronize with each other by placing locks on tables, pages, or rows to prevent simultaneous modification of the same data object. Each lock is associated with a descriptor in the global shared memory Lock Table.

Every time a lock is acquired, a free entry in the table is allocated for it. The entry is freed for reuse when the lock is released. If the maximum number of locks is reached, a new lock cannot be acquired until one of the locks is released and a free entry becomes available. A lock descriptor may be on one of two waiting lists: 1) a list of locks placed by that process; or 2) a list of shared locks for the same object.

4.4.1.3 Dbspace Table

The dbspace is the largest logical unit of disk space. For each dbspace, there is a descriptor in the global shared memory Dbspace Table. This table, like all the tables in global shared memory, is fixed in size, but unlike the User Process and Lock Tables, the information kept in the Dbspace Table is persistent and brought into global shared memory from disk at initialization time. Every time a new dbspace is created, a free entry in the table is allocated for it. The entry is freed for reuse when the dbspace is dropped. If the maximum number of dbspaces is reached, a new dbspace cannot be created until one of the current dbspaces is deleted and a free entry becomes available.

4.4.1.4 The Chunk and Mirror Chunk Tables

Chunks are discussed in detail on page 33, "Chunk Management". For each chunk, there is a descriptor in the global shared memory Chunk Table. The information in the Chunk Table is persistent and brought in from disk when INFORMIX-OnLine/Secure is initialized. Every time a new chunk is added to a dbspace, a free entry in the Chunk Table is allocated for it and the entry is placed on the linked list of chunks for the corresponding dbspace. The entry is freed for reuse when the chunk is removed. If the maximum number of chunks is reached, a new chunk cannot be allocated until one of the current chunks is removed and a free entry becomes available. Each chunk descriptor must be on exactly one linked list of chunks for a particular dbspace.

A dbspace can be mirrored so that a media failure can be recovered gracefully. Within a mirrored dbspace, data are replicated on mirror chunks, which are typically on a different device from the primary data chunks. For each mirror chunk, there is a descriptor in the Mirror Chunk Table that is analagous to the descriptor in the Chunk Table.

4.4.1.5 Tblspace Table

A tblspace stores rows and indexes of a table. For each tblspace opened by any process, there is an entry in the Tblspace Table. This information is brought into memory from disk when the first RDBMS Kernel process opens the tblspace. Every time a new tblspace is built to create a table, or an RDBMS Kernel process opens an existing tblspace for the first time, a free entry in the Tblspace Table is allocated for that tblspace. As with the other global shared memory tables, if a free entry is not available, a new tblspace cannot be built nor can an existing (previously closed) tblspace be opened until an RDBMS Kernel process closes a tblspace and frees an entry in the global shared memory Tblspace Table.

4.4.1.6 Buffer Table and Buffer Pool

A large pool of buffers is maintained in global shared memory to cache data pages and index pages read from disk. Each buffer is the same size as a physical page so that a cached page can be stored in it. The cache, or buffer pool, is available to all active RDBMS Kernel processes. A RDBMS Kernel process seeking a particular data page may find it already in the cache as a result of a previous I/O initiated by itself or another RDBMS Kernel process, and thereby avoid an unnecessary disk read operation. Caching disk pages allows for faster I/O for those pages already in the cache. Associated with *each* buffer is a descriptor in the Buffer Table.

Every time a page is read from disk into the global shared memory cache, a buffer descriptor in the Buffer Table is allocated for it. Any access to the buffer itself must go through the buffer descriptor. A latch is used to ensure exclusive access to the buffer descriptor so that no two processes can manipulate it at the same time and interfere with each other. The status of the buffer is maintained by a flag in the descriptor, indicating whether the buffer contains a data page, a dirty data page, and whether it is on any LRU list.

To facilitate quick access, buffer descriptors are hashed into hash buckets based on the logical page address of the disk page stored in the associated buffer. A linked list of buffer descriptors is maintained within each bucket for those that are hashed to the same bucket. To find a buffer that contains a given page, the hash function is applied to the physical page number and the result is used as an index to the particular hash bucket. The linked list of buffer descriptors in the bucket is searched until the desired page is found.

If a page cannot be found in the cache, a disk I/O is performed to read the desired page into a selected buffer. An LRU list is maintained by linking the buffer descriptors ordered by the last time the associated buffer was accessed. The buffer descriptor for the most recently used buffer is at the head of the list, while the descriptor for the least recently used buffer is at the tail. When a cache hit occurs (the desired page is found in the cache), the corresponding buffer descriptor is removed from the LRU list. After the buffer is used and released, its descriptor is put back at the head of the LRU list. If one of the cached pages has to be replaced to make room for another page about to be read in from the disk, the buffer whose buffer descriptor is at the tail of the LRU list is selected. It is possible that the LRU list can become a bottleneck when there is heavy contention for buffers. To resolve this problem, several LRU lists are available.

If a cached page has been modified since the last time it was flushed (called a dirty page), the dirty page must be written back to disk before it can be replaced. To increase the probability that the replacement algorithm finds a clean candidate that does not require another disk I/O, cached pages are flushed to disk in groups. Buffered pages are partitioned into these groups based on their physical locations on disk. Physically near-by pages are grouped together in the same *near list*. If a buffer needs to be written back to disk, all dirty pages in the same near list are flushed as well. The near list is maintained in the buffer descriptors. Note the near list only applies in cases where chunks are on raw devices; if chucks are cooked files then the OS determines the buffering scheme.

To maximize concurrency, multiple processes are allowed to access the same cached page for reading. Only a single process at a time is allowed to access the page in a buffer for writing. Three types of locks are used to maintain this concurrency control on buffers: share lock, exclusive lock, and update lock. A detailed discussion of locks can be found on page 57, "Lock Management".

This buffer scheme is not used for BLOBs residing in BLOBspaces. I/O for in-tblspace BLOBs, on the other hand, does use this buffering scheme. However, for a write operation, the acquired buffer is released as soon as it is flushed to disk; for a read operation, the buffer is released immediately after the data is transfered to the user buffer. For off-tblspace BLOBs (BLOBspace resident), physical I/O is invoked directly so that even this temporary use of the cache is bypassed. BLOBspace BLOB data is temporarily kept in a process buffer while the I/O operation is in progress.

4.4.1.7 Logical Log Buffer

The global shared memory contains three logical log buffers when logical logging is enabled. The logical logs contain a record of all changes to the databases. It is then possible to rollback these changes using the logical logs. The size of a logical log buffer is configurable, and typically is much larger than a disk page. A logical log buffer is written from memory to disk when the log is full, when a transaction is committed, or when a checkpoint takes place (see page 56, "Transaction Management" for a discussion of logical logging).

4.4.1.8 Physical Log Buffer

The physical log contains before-images of disk pages which are used to recover from a system crash. In general, a write to a buffer that will result in an update to a disk page is preceded by a write of its before-image to the physical log. Two physical log buffers are contained in global shared memory and are rotated so that while one physical log is being used the other can be flushed to disk. The size of the physical log buffers is configurable but is typically large enough to contain several disk pages.

4.4.1.9 Page Cleaner Table

Page cleaners are daemon processes whose job it is to flush dirty buffer pages to disk. Generally, there is one page cleaner for each disk spindle. This allows multiple page cleaners to work in parallel, each concentrating on one device. The page cleaner table in global shared memory is part of the Flush Control structure and

SECRET row	Jones	$60,000	IRS Agent
UNCLASSIFIED row	Smith	$35,000	Carpenter
SECRET row	Jackson	$48,000	FBI Agent
TOP SECRET row	Clinton	$95,000	President

Figure 4.7. What the user sees

always has 32 slots available. Thus, the number of page cleaners is configurable but limited to 32. If zero page cleaners are configured, the master daemon does the job of the page cleaners. Otherwise, the master daemon is responsible for assigning work to the page cleaners. See page 67, "Daemons" for a detailed discussion of daemons.

4.4.2 Bundles

A bundle is a realization within the RSAM process of a table. Multi-level tables can result in the existence of the same row at multiple sensitivity labels; this is called *polyinstantiation*. INFORMIX-OnLine/Secure does not intentially use polyinstantiation, however, it is possible for polyinstantiation to occur if, for example, a process at a higher sensitivity label reads a row with a lower sensitivity label and subsequently inserts the row back in the table. The row inherits the higher sensitivity label because writing down is not allowed. Thus, the row now exists at two sensitivity labels. If a user does not specify a particular sensitivity label when selecting a row, all lower-level versions of a polyinstantiated row are retrieved. The clean-up procedure for polyinstantiated rows is site-dependent.

INFORMIX-OnLine/Secure uses the *bundle* abstraction to implement mandatory access control. Conceptually, a table within a database is seen by the user as a multi-level object as shown in Figure 4.7. In actuality, a tblspace is created for each sensitivity label in the table. A special tblspace is created to locate the tblspaces at each sensitivity label. This special tblspace is called a *bndlspace*. The bundle consists of the bndlspace and its subordinate tblspaces as shown in Figure 4.8. Tables are accessed via the bundle abstraction. In the same way that each RSAM process maintains the root data structure and its subordinate structures, RSAM must also maintain a *root bundle data structure* and its subordinate structures. All of these structures are created, owned, and maintained by RSAM for the user on whose behalf RSAM is running.

4.4.3 The System Catalog

The system catalog is a collection, or catalog, of system tables that describe the structure of a database. Each table in the system catalog contains specific information about a structural element in the database, i.e., users, tables, views, columns, rows, indexes, synonyms, and constraints. The system catalog is automatically generated every time a new database is created and is stored as database tables in the dbspace in which the database is created. The system catalog is at the same sensitivity label as the database.

During the process of opening the database, RSAM initializes an open_db structure for the database as described in the previous section. The logical address of each table in the database's system catalog is copied from the **systables** table to an array in the open_db structure. The system catalog consists of the following tables:

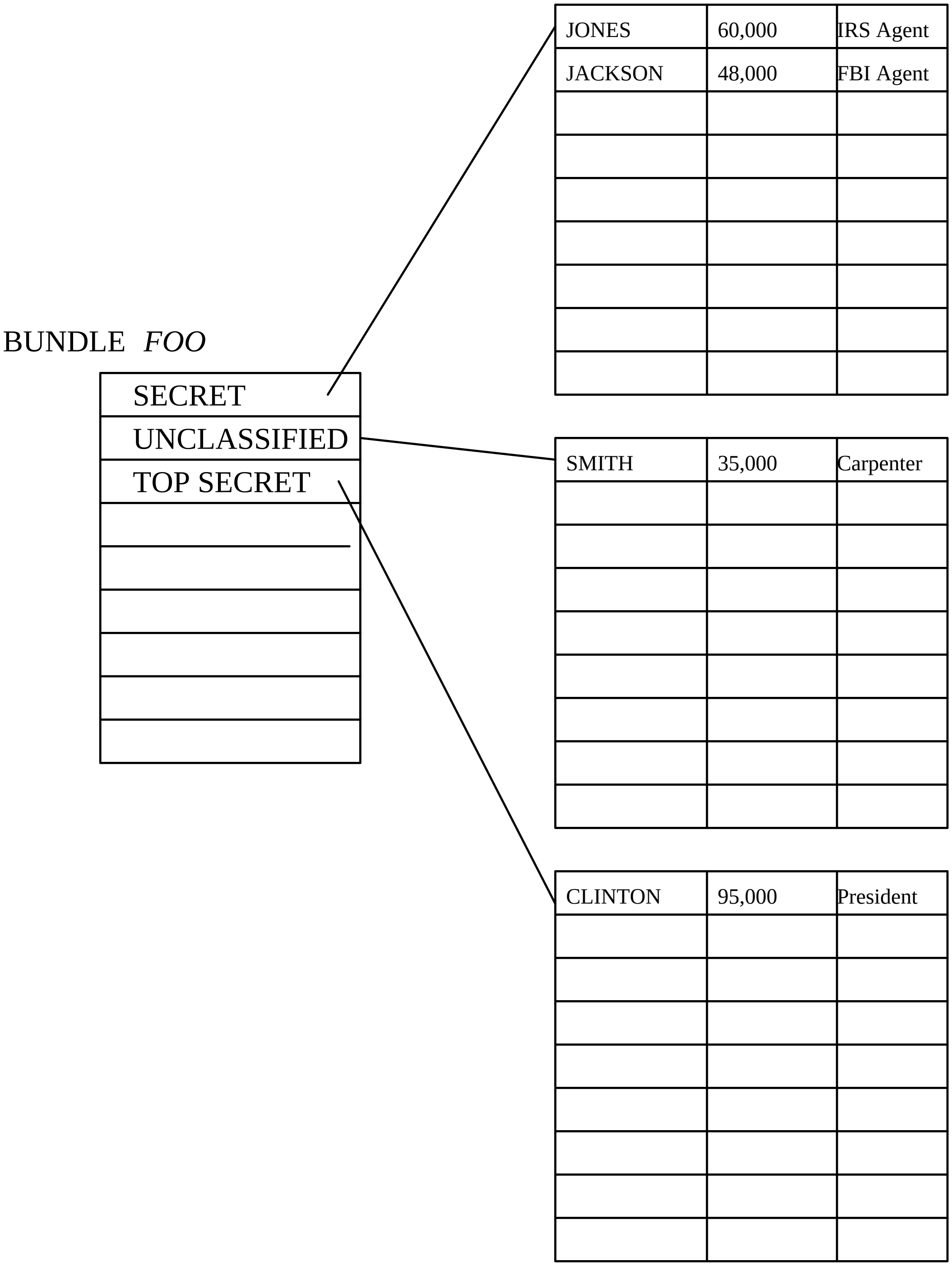


Figure 4.8. The bundle

systables - Describes the tables in the database, including user tables and system tables,

syscolumns - Describes the columns in the tables,

sysindexes - Describes indexes on tables,

systabauth - Identifies table level DAC privileges,

syscolauth - Identifies table level DAC privileges specific to columns,

sysdepend - Describes how views depend on tables or other views,

sysusers - Identifies database level DAC privileges,

sysviews - Defines views,

sysconstraints - Records constraints placed on database tables, and

syssyntables - Used for mapping synonyms and tables.

4.4.3.1 Systables

The **systables** is the most important table in the system catalog. It maintains information used to locate all of the tables in the database - both system catalog tables and regular database tables. Each system catalog table has a tabid, or table identifier, within the database between 1 and 11; each database table has a tabid greater than 100. Thus, the tabid distinguishes system catalog tables from database tables. In general, the tabid, rather than the table name, is used to refer to a specific table within a database.

Each row of **systables** identifies a table defined in the database. Whenever a new table, view, or synonym is created in the database, a new entry is automatically added to the **systable** table. Each new entry contains information such as the type of table (table, view, synonym), name of the table, the owner, the number of columns and rows, and the creation date. The logical address of the tblspace where the table resides is also stored in each **systables** row as well as the minimum sensitivity label of data which can be stored in the table. The minimum sensitivity label also serves as the sensitivity label of each row in **systables**.

4.4.3.2 Syscolumns

The **syscolumns** table keeps track of columns in all the tables in the database, including the system catalog tables. Each row contains a column name, the tabid of the table to which the column belongs, the column number within the table, the type of column, and the physical length of the column. Column numbers are sequentially assigned by the system from left to right within each table. There are 14 different column types including character, integer, date, byte. The sensitivity label of each row in the **syscolumn** table is the same as the table to which the column belongs. The row describing a column of a particular table is updated whenever there is a schema change on any column of that table. These rows are indexed by the tabid and column number.

4.4.3.3 Sysindexes

This table describes each index created on tables in the database. Each row contains an index name, the owner of the index, the tabid of the table, the index type, clustering information, and the column numbers of the columns of the index. The sensitivity label is the same as the table on which the index is created. Creating a new index in the databases causes a new entry to be added to **sysindex**. Dropping an index in the database causes that entry to be deleted from the **sysindexes** catalog. When an index is changed from clustered to non-clustered, or vice versa, the entry in **sysindexes** is updated. Clustered indexing means that all rows that have the same index are stored contiguously. This is only meaningful when a table is altered. When a row is inserted, clustering is not maintained.

4.4.3.4 Systabauth

The **systabauth** records the DAC table level privileges for each table in the database. Each row contains the grantor and grantee of the privileges, the tabid of the table, and the type of privilege. There are six types of privilege that can be defined at the table level: **(a)lter**, **inde(x)**, **(d)elete**, **(i)nsert**, **(u)pdate**, and **(s)elect**. These privileges are represented in each entry of the **systabauth** table by a seven character code where each character represents one of the privileges as follows: su*idxa. If a privilege has been granted, the appropriate letter will appear in the character code. A hyphen indicates that a particular privilege has not been granted. If the privilege is in uppercase, the user granted this privilege can also grant it to others; lowercase means the user does not have the grant option. If privileges have been granted or revoked on a column, an asterisk will appear as the third character in the character code. It should be noted that each row also contains a sensitivity label that is the same as that of the table on which the privilege has been granted (see page 75, "Discretionary Access Control" for more information about DAC).

4.4.3.5 Syscolauth

This table is used to support DAC on tables in the database to the granularity of a column. Each row contains the grantor and grantee of the privilege, the tabid of the table on which the privilege has been granted, the number of the column within the table, and a two character code which identifies the type of privilege. There are only two types of table level privileges that apply to columns - **(u)pdate** and **(s)elect**. They are stored within each row of the **syscolauth** table in the same manner as the seven character code is stored in the **systabauth** table. Each row also contains the sensitivity label of that row which is the same as the sensitivity label of the table to which the column belongs.

4.4.3.6 Sysdepend

This table is used to describe how views depend on other tables or views. The tables or views on which a view is defined are called *base objects*. The view itself is called a *dependent object*. The **sysdepend** table maintains the base object type (table or view) and its tabid, the dependent object type (the view) and its table identifier, and a sensitivity label of the view. This table gets updated when a new view is created or an existing one is dropped. A user who has **alter** privilege on a base object can cause a view to be dropped.

4.4.3.7 Sysusers

This table keeps database level privileges of users. There are three types of database level privileges: **dba**, **resource**, and **connect**. The **sysusers** table contains the user ID and user type, indicating the database level privileges possessed by the user. This table is used by RSAM to validate a user's request to access the database. Information in this table gets updated whenever grants and revocations of privileges at the database level take place.

4.4.3.8 Sysviews

This table keeps information about views in the database. It is used together with the **sysdepend** table to manipulate views. Each row stores that portion of SELECT SQL statement that defines a view. For each view the SELECT statement of a view may use up several rows in **sysviews**. The statement is divided into fixed-length sections. Each section is preceded with the tabid of the view and the sequence number of the section where it is stored in the row. Also, the sensitivity label of the row is the same as that of the view. This table is updated when a new view is defined or an existing view is modified due to changes in the base objects.

4.4.3.9 Sysconstraints

This table keeps information on all constraints specified for tables in the database. For each constraint the internal constraint number, constraint name and the owner of the constraint is maintained. In addition, the tabid of the table for which the constraint is specified, name of index created is also specified if it is a unique constraint. Note that the sensitivity label of the row in the **sysconstraints** table is the same as the sensitivity label of the table on which the constraint is created. A new entry is inserted into this table when a new constraint is created for a table, an entry is deleted when a constraint is dropped. A user who has the **alter** privilege on a table can specify constraints.

4.4.3.10 Syssyntables

This table keeps information on how synonyms relate to their base tables or views. The information stored in **syssyntable** includes: tabid of synonym, database name of the base object (NULL if the same database), server name[6] for that database, name of the base object, owner of the base object, tabid of the base object. Note that the sensitivity label of the row is the same as the sensitivity label of the synonym.

When a new table is created as a result of altering a table, entries related to synonyms which apply to the original table get updated with the new table identifier. A user who has alter privileges on a base object can cause changes on this particular table.

[6] The server name is nothing more than a name in the evaluated configuration and is used to generate the key for global shared memory. In future versions of INFORMIX-OnLine/Secure, it will be used to identify a remote DBMS server in a distributed environment.

dbspace number 8 bits	tblspace number 24 bits

Figure 4.9. Tblspace logical address

4.4.4 Database and Table Management

This section describes how RSAM services user requests to manipulate entities of the RDBMS, specifically databases and tables. A discussion of how RSAM handles operations on rows is found on page 53, "Row Management". On page 55, "BLOB Management" the manner in which RSAM services requests on BLOBs is discussed. The following operations are described:

- Database Operations: create, drop, open, close;
- Table Operations: create, drop, open, close, alter, rename;
- Row Operations: insert, delete, read, update;
- BLOB Operations: create, delete, open, close, read, copy.

4.4.4.1 Creating and Opening a Table

This section first describes the process of creating a table. In order to create a table, the database that is to contain the table has to exist and be open. In addition, the table must have a unique name. The process of creating and opening a database is described on page 52, "Creating and Opening a Database".

Since tables are implemented as bundles, a bundle must be created with the parameters specified for the new table. To create a bundle two tblspaces must be created: one for the bndlspace and one to store the rows at the sensitivity label at which the table is being created[7].

In order to create these tblspaces, an entry must be allocated for each one in the global shared memory Tblspace Table as described on page 42, "Tblspace Table". In addition, entry for each tblspace must be added in the tblspace-tblspace of the dbspace where the tblspaces are to reside. A row is then added to the bndlspace to indicate the sensitivity label of the user session and the logical address of the tblspace that is to contain the rows at that sensitivity label. The appropriate tables of the system catalog must also be updated to reflect the addition of the new table. A new table ID, called a tabid, is generated for the new table. The tblspace number of the bndlspace and the tabid is returned to the calling process upon successful creation of the table.

In order to open a table, the database that contains that table must be open. RSAM uses the name of the table to retrieve the logical address of the table from **systables**. RSAM also performs MAC and DAC checks using the information retrieved from **systables** and **systabauth**. The table open operation will fail at this point if the calling process does not have MAC and DAC access (see page 79, "Mandatory Access Control" and page 75, "Discretionary Access Control").

Since tables are implemented as a bundle, the logical address of the table is actually the logical address of the bundle, i.e., the bndlspace. The logical address of a tblspace is 32 bits as shown in Figure 4.9. The high-

[7] The sensitivity label of the new table is equal to that of the user session creating the table.

order 8 bits are the dbspace number where the tblspace is located. Given the dbspace number, the physical address of the root chunk of the dbspace can be retrieved from the global shared memory Dbspace Table. The root chunk of a dbspace contains the tblspace-tblspace which describes all the tblspaces in the dbspace (see page 33, "Dbspace Management"). The low-order 24 bits of the tblspace logical address indicates the page number in the tblspace-tblspace that holds the descriptor for the bndlspace. The descriptor gives the physical address, with the unique chunk number encoded in the high-order 12 bits. Given the chunk number, RSAM looks in the open chunk table to see if the chunk is already opened by this process. If it is, RSAM can retrieve the UNIX file descriptor (fd) which is stored in the open chunk table. If not, RSAM searches the global shared memory Chunk Table to retrieve the UNIX path name for the chunk and then issues a UNIX *open* call to get a UNIX fd for the chunk. An entry is then added to the open chunk table storing this fd.

In addition, an entry must be added to the open table for this open instance of the tblspace. If this is the first open instance for this process on the tblspace, an entry must also be added to the file table, otherwise the open count for this tblspace is incremented.

Since a bundle is being opened, the root bundle data structure, created at session startup, and its subordinate structures must also be initialized. An entry must be added to the open bundle table for this open instance on the bundle. If this is the first open on the bundle, an entry must also be added to the bundle file table, otherwise the open count for this bundle is incremented. An entry is added to the isfd-to-bfd lookup table at the same isfd as the bndlspace tblspace in the open table which contains the bfd in the open bundle table for the bundle.

At this point, RSAM examines every row in the bndlspace and performs a MAC check. For each row whose sensitivity label is dominated by the sensitivity label of the user session, RSAM opens the associated tblspace. The bndlspace row contains the logical address of the tblspace so the process of opening each tblspace that meets the MAC criteria is the same as that previously described. The only difference is that the open bundle table, bundle file table, and isfd-to-bfd lookup table have already been initialized. For each tblspace that meets the MAC criteria, a structure is added to the visible tblspace list which contains the isfd for each open tblspace. For subsequent access requests on the table at a particular sensitivity label, RSAM uses the isfd found in the appropriate visible tblspace list structure to access the open tblspace.

4.4.4.2 Dropping and Closing a Table

Dropping a table is the opposite of creating a table. In order to drop a table, the table cannot be in use by any other users. RSAM also performs MAC and DAC checks using the information retrieved from **systables**. The drop operation will fail at this point if any of these three conditions are not met.

Since a table is implemented as a bundle, the bndlspace number of the bundle is read from the **systables** entry and the bundle is opened. The bndlspace is locked in exclusive mode and all the labeled tblspaces indicated by the bundle are opened. Subsequently, each labeled tblspace is dropped and the bndlspace is dropped as the last tblspace in the bundle. The open table entry and the file table entry are set to all zeroes and the disk space used for the table is marked available.

Closing a table requires no special MAC or DAC privilege; if the user could access the table, then the user has sufficient MAC and DAC privilege to close the table. All locks on the table will first be released if appropriate based on the type of lock. The bndlspace for the table must also be closed. The open count in the bundle file table for the bundle and the open count in the file table for the bndlspace and the tblspace must be decremented. Appropriate entries in the open table and the open bundle table are cleared.

4.4.4.3 Altering a Table

The alter table operation changes the schema, which is the description of the rows in the table, by changing the attributes of the rows in the table. The following changes can be made to the schema:

- Add a column,
- Delete a column,
- Modify the data type of a column,
- Add a unique constraint to a column or a composite list of columns[8],
- Drop a constraint associated with a column or a composite list of columns,
- Modify the size of the next extent, and
- Change the locking level for the table.

If RSAM determines that the user has the required MAC and DAC access to alter the schema (see page 79, "Mandatory Access Control" and page 75, "Discretionary Access Control"), RSAM attempts to exclusively lock the table. The alter operation will fail if any other user has access to the table.

If the alter operation does not include the addition or deletion of a column, or a modification in the type or length of a column, RSAM simply updates the **syscolumns**, **sysindexes**, and **sysconstraints** tables of the system catalog accordingly after performing the alter. These types of modifications do not cause the deletion or addition of entries in **systables** but instead update the existing entry to reflect the changes.

If the alter operation involves any column addition, deletion, data type or length changes, a new table is built incorporating the old table with the changes. Since a table is implemented as a bundle, this process includes creating a new set of tblspaces in the dbspace in which the original tblspaces reside and updating information on the size of the next extents. A new table identifier is assigned and the new table is locked in the same mode as the original table. A new entry is added to **systables** with the new table identifier. The entries related to the original table in **systabauth**, **syscolauth**, **sysdepend**, and **syssyntables** are modified with the new table identifier. After the new table is built and the system catalog is successfully updated, the bundle which implemented the original table is dropped and all entries related to the original table identifier are deleted from the system catalog.

The version number is updated every time the table is altered. If a new table is built as a result of an alter operation, its version number is based on the version number of the original table, and updated to a new number according to the changes that were made. Within an alter operation, the version number of the table increments by one for each addition or deletion of a constraint on the table. The version number is also incremented once for the alter operation itself.

4.4.4.4 Renaming a Table

A table rename operation changes the name of a user table. If RSAM determines that the user has the required MAC and DAC access to rename the table (see page 79, "Mandatory Access Control" and page 75, "Discretionary Access Control"), RSAM attempts to lock the table and update the name in the **systables** table of the system catalog.

[8] There are two types of constraints: a unique constraint which requires that the value in a column be unique across the table; and a *not NULL* constraint which requires that the column always have a value, i.e., it cannot be NULL.

4.4.4.5 Creating and Opening a Database

This section describes how databases are created and opened. Databases are the largest logical unit within the RDBMS. In order to maintain all of the databases in the RDBMS, a single table is stored in the root dbspace called the *database tblspace*[9]. The database tblspace contains one entry for each database in the RDBMS.

When a database is created, an entry is allocated in the database tblspace for the new database. The sensitivity label is set to the sensitivity label of the process which initiated the creation of the database. The database and its associated system catalog is created and stored in the dbspace specified by the user (the root dbspace is the default if no dbspace is specified). Note that if the user specifies a dbspace, it must have an entry in the global shared memory Dbspace Table (see page 41, "Dbspace Table"). All tables of the system catalog are created with the same sensitivity label as the database and are owned by the user ID informix. The system catalog is initialized as follows:

- The **sysusers** table contains an entry for the creator of the database and indicates **dba** privilege for that user.
- The **systabauth** table is initialized so that all users have the **select** privilege on the system catalog.
- The **syscolumns** table is initialized with information about the columns for all tables of the system catalog.
- The **sysindexes** table is initialized with any index information for all tables of the system catalog.
- The **systables** table is initialized with the tblspace number and the logical address of the tblspace for each table of the system catalog.

The process of creating a new database is an atomic transaction, i.e., the creation is not performed unless every step in the initialization process is successfully completed. Once this transaction completes successfully, the database is opened and a success message is returned to the calling process. It should be noted that only one database may be the current open database for a process at a time. The database creation process will fail if another database is already opened as the current database for the process.

A database can be opened for a user process only if an entry exists for that database in the database tblspace, the user has DAC and MAC access, the requested locking mode can be set for the database, and the user process doesn't currently have any other database opened.

Provided these criteria are met, the following steps are taken in order to open the database. The system catalog **systables** table is opened with the requested locking mode. The locking mode is also set in the database tblspace entry. Since **systables** is a table, it must be opened as described on page 49, "Creating and Opening a Table". Once **systables** is opened, logical addresses of all the tables in the system catalog are copied into the RSAM *open_db* structure for future use by RSAM in accessing the database. The database level privileges are copied from **sysusers** and stored in the *open_db* structure for subsequent DAC checks (see page 75, "Discretionary Access Control"). At this point the database becomes the current database and a success message is returned to the user.

[9] The database tblspace is not part of any database.

4.4.4.6 Closing and Dropping a Database

To close a database, RSAM closes **systables** as described on page 49, "Creating and Opening a Table" as well as all other database tables. All locks held by the process are released and all linked-list entries in the in-core dictionary are released. The *open_db* structure is cleared.

Dropping a database is the opposite of creating a database. In order to drop a database, no other users may have the database opened, the user must have MAC and DAC access to perform the drop operation, and the database must not be in use by any other users. If all three conditions are true, each user table is dropped from the database in the sequential order as recorded in **systables** starting with the first entry (see the discussion of the drop table operation on page 50, "Dropping and Closing a Table"). After the user tables are dropped, all tables in the system catalog are dropped in the same manner with **systables** dropped last. Finally, the entry for the database in the database tblspace is dropped.

4.4.5 Row Management

This section discusses insertion and deletion of a row, reading a row, and modifying or updating a row. A row is stored in a tblspace. For all row operations, RSAM searches the visible tblspace list to locate the tblspace at the user's sensitivity label, or the set of tblspaces dominated by the user's sensitivity label, depending on the particular operation.

4.4.5.1 Inserting a Row

To insert a new row in a table, the requesting process must have opened the table and supplied a new record to be inserted. The user has all the necessary MAC permissions to insert the row if both the database and table were successfully opened. RSAM performs a DAC check before the insertion is allowed (see page 75, "Discretionary Access Control").

Data pages store the rows for tblspaces (see page 34, "Page Management" for a detailed discussion of page structure). RSAM determines if the row can be stored in a single page and the size of the required slot. If the row is longer than a page, a forward pointer is needed and an entire page is devoted as the home page of the new row. A page with enough room is located within the tblspace that has the same sensitivity label as that of the requesting process. Locating a suitable page also serves to establish the rowid of the new row which consists of the logical page number and the slot number. The new record is copied into the available slot space in the home page. If a remainder page is needed, the forward pointer in the home page cannot be filled until the first remainder page is found. As many remainder pages as needed are located within the tblspace.

4.4.5.2 Deleting a Row

The requesting process can delete either the current row, a row identified by a particular rowid and sensitivity label, or a row whose field(s) match a user-supplied key value. In order to delete a row, the user must have opened both the database and the table. RSAM locates the target row by searching the visible tblspace list, as previously described. If the row is not in the tblspace with a sensitivity label equal to that of the requesting process, the delete operation will fail. Once the row is successfully located, RSAM ensures that the row is not in use by another user, performs a DAC check, and places an exclusive lock on it. The row is deleted by adjusting the slot table and deallocating the slot (see page 34, "Page Management" for a detailed discussion of slots). If the target row is longer than the home page, the forward pointer is followed to each remainder page to mark the deletion in the associated slot table. If the row deleted is the last row in a labeled tblspace, the labeled tblspace remains in the bundle even though it is empty.

4.4.5.3 Reading a Row

When a row is to be retrieved from a table, the requesting process must provide information directing RSAM on how to locate the target row. In order to retrieve a row for reading, the user must have opened both the database and the table. Once the row is located, RSAM performs a DAC check and sets either a visible or invisible lock (see page 57, "Concurrency Management" for a detailed discussion of locks and isolation levels). If the sensitivity label of the requesting process equals that of the tblspace that contains the row and the isolation level is not dirty read, a visible lock is set; if the sensitivity label of the requesting process dominates that of the tblspace that contains the row only a dirty read can take place and an invisible lock is set.

When the table is opened, if the requesting process does not have **select** privilege on the table, a *read mask* is constructed and stored in the open table. The read mask is the same size as the row. Bits are set in the read mask for each column that is readable by the process and cleared for unreadable columns. The read mask is used by RSAM to construct the requested row. Beginning on the home page of the row, RSAM reads the row following any forward pointers and using the read mask until the entire row is assembled. The row is then returned to the user.

4.4.5.4 Updating a Row

The requesting process can update either the current row, a row identified by a particular rowid and sensitivity label, or a row whose column(s) match a user-supplied key value. In order to update a row, the user must have opened both the database and the table. RSAM locates the target row, performs a DAC check, and ensures that the row does not have any visible locks placed on it by another process. An error is returned to the user if the sensitivity label of the tblspace containing the row is not equal to the sensitivity label of the requesting process. Once the target row is successfully located, the row is exclusively locked.

The requesting process may be able to update all columns in an existing row, or only some of the columns. When the table is opened by the requesting process, a *write mask* is constructed and stored in the open table. The write mask is the same size as the row. Bits are set in the write mask for each column that is updatable by the process and cleared for non-updatable columns. The write mask is used by RSAM to assemble the new row. RSAM then determines if the updated row can be stored in its entirety in the home page. If there is not enough room, part of the row has to go to a remainder page. However, a portion of the row is left in the home page to keep the rowid unchanged. If the old row was longer than the updated row, trailing remainder pages or portions of pages are appropriately released. If the old row was shorter than the updated row, new remainder pages are allocated.

4.4.6 BLOB Management

INFORMIX-OnLine/Secure supports multi-media databases which can contain Binary Large Objects (BLOBs). A user can create a BLOB for later inclusion as part of a row in a database[10]. BLOBs are created from an operating system file by copying the file into either in-tblspace BLOB pages or BLOBspace BLOB pages. Typically, BLOBs are stored in the tblspace where the target row is located when the BLOB is small. A special dbspace called a BLOBspace is used to handle large BLOBs. On page 37, "BLOBspace Management" the details of disk management of BLOBs is discussed. This section discusses how RSAM services user requests to manipulate BLOBs.

4.4.6.1 Creating and Opening a BLOB

After a BLOB has been created either in the target tblspace or in a BLOBspace, the BLOB still does not have any of the necessary linkages to a row. The BLOB must be associated with a row which requires that the database and table be open. RSAM performs the necessary MAC and DAC checks prior to opening the database and table. When the tblspace is opened, the open table structure for that tblspace contains a pointer to the open BLOB table which contains one BLOB descriptor for each BLOB column in the tblspace. This descriptor points to an entry in the special column descriptors array which describes the characteristics of VARCHAR and BLOB columns. The BLOB descriptor also points to the tuple BLOB structure which contains the column offset, isfd, tblspace number and some additional information about the location of the BLOB. If a BLOB has not yet been associated with a row, the pointer to the tuple BLOB structure will be NULL. An existing row with a NULL BLOB column can be updated to associate the BLOB with that row. Alternatively, a new row can be inserted into the table. If the BLOB is to be associated with a new row, the row is not written until the tuple BLOB structure is initialized to point to the BLOB which is entered in the BLOB column. RSAM creates the tuple BLOB structure indicating the column offset, ifsd, and tblspace number and initializes the BLOB descriptor to point to the tuple BLOB structure.

A user who requires access to a BLOB must first open the BLOB. In order to open a BLOB, RSAM ensures that no other user has an exclusive lock on the BLOB. In addition, only one BLOB can be open by the user process at a time no matter how many tblspaces the user has open. If all of these conditions are met, RSAM sets the BLOB pointer in the root data structure to point to requested BLOB in the open BLOB table.

4.4.6.2 Deleting and Closing a BLOB

A BLOB can only be deleted when a row that contains a BLOB is deleted or a column which is a BLOB column is updated. There is no access to BLOBs that are not part of a row.

Closing a BLOB relinquishes access to an open BLOB, completing any pending operations. Memory structures are restored to a state similar to the state they were in before the BLOB was opened. If a close operation is issued while the BLOB is being created anything in the BLOB write buffer is written to the BLOB before closing. If the BLOB is open for read when the close BLOB operation is issued, an end-of-access indication is left in the open BLOB table structure.

[10] BLOBs are not considered MAC objects but are part of a row. The sensitivity label of the BLOB is that of the row that contains it.

4.4.6.3 Reading a BLOB

Reading a BLOB results in a specified number of bytes read from a BLOB, or the number of bytes until the end of the BLOB, whichever is smaller. BLOB reading is a sequential operation. For in-tblspace BLOBs, if no buffer was previously allocated for reading BLOBs, RSAM allocates a buffer. For BLOBspace BLOBs, special buffers are allocated in RSAM local space. Data is transferred until the number of bytes has been read. RSAM updates the access information which is stored in the open BLOB table structure for the BLOB and in the root data structure for the next read operation.

4.4.6.4 Copying a BLOB

A BLOB copy operation creates a new copy of an existing BLOB. This is the only situation in which two BLOBs can be open concurrently. First, RSAM performs a DAC and MAC check on the rows that contain the BLOB to be copied, called the source BLOB. The source BLOB is then copied to a temporary place in RSAM local space and the BLOB pointer in the root data structure is set to NULL. The target BLOB is created, a new tuple BLOB structure is allocated and initialized and a new entry is allocated in the open BLOBs table and initialized to point to the new tuple BLOB structure. BLOB read and write operations are repeated until all designated portions of the source BLOB have been copied to the target BLOB. The BLOB pointer in the root data structure is set to point to the new entry in the open BLOB table and the target BLOB becomes the current BLOB.

4.4.7 Transaction Management

INFORMIX-OnLine/Secure is designed with fault-tolerance as a specific design goal. Therefore, the system is designed to ensure that the database is always in a logically consistent state or can be restored to a logically consistent state even in the event of a failure of the system. A logically consistent state is defined by the use of *transactions*. A transaction is an operation that starts with a *begin work* statement, does whatever changes are appropriate, and then finishes with a *commit work*. Many pieces of information in a database can change within a single transaction. INFORMIX-OnLine/Secure is responsible for ensuring that all transactions are logically atomic operations. Any time before issuing the commit work, all of the changes since the begin work can be completely undone via the *rollback work* statement. INFORMIX-OnLine/Secure ensures that all transactions are either completely undone or completely finished.

4.4.7.1 Transaction Logging

Transaction logging (also called logical logging) provides the ability to save the "before" images of rows involved in a user transaction until all the operations specified in that transaction are complete. The primary goal of transaction logging is to support transaction rollback and rollforward operations. The transaction can be rolled back and started again if a problem occurs. The *logical log* contains one record for each change operation performed during the transaction. Thus, for a single transaction, the logical log contains a *begin work* record, a set of change records, and a *commit work* record. All logical log records consist of a log header which contains the length of the record, the record type, the transaction ID, and information for finding previous log records within the same transaction. Log records within the same transaction are linked so that transactions for different users can be distinguished. In addition to the log header, the log record contains additional information that is dependent on the record type. Logical log records are stored on pages which comprise the logical log.

To further ensure that the system remains in a consistent state, a Compensation Log Record (CLR) is used for logging during rollback in the event of a system failure during the rollback operation. Like other log records, it contains a header and information on what is being done. However, the information in the log header has the log unique ID and log position of the log that is being undone rather than the address of the previous log record in the transaction.

4.4.7.2 Logical Log Operations

The DBSA must be at the **DataLo + IX_DBSA** sensitivity label to perform any logical log operation. Only the DBSA can add or delete a logical log. The DBSA adds another logical log in the event that the existing logical log is filling up too quickly. Only logs that are not in use and not locked can be deleted. Once deleted, the log is returned to the free pool.

The DBSA can also back up the logical log to off-line storage. This provides the ability to recover from failures that may have corrupted logical logs as well as data in the raw device. This allows recoveries without a complete restore from the archive.

The logical logging mode specifies whether transaction logging is enabled as well as the type of logging to be performed. The DBSA can specify a transaction logging mode for a database. Only the DBSA can change the transaction logging mode and the database must not be in use by any other users at the time of the change. The system must be in the quiescent state in order to change the transaction logging mode.

4.4.8 Concurrency Management

Concurrency control includes lock management and data structure integrity. This section discusses both types of concurrency control.

4.4.8.1 Lock Management

Locks are the primary means of ensuring data integrity throughout INFORMIX-OnLine/Secure. Currently active locks are managed in the Lock Table and the User Process Table in global shared memory (see page 40, "Global Shared Memory" for a detailed discussion of these data structures). INFORMIX-OnLine/Secure supports several distinct types of locks:

- Shared Lock - allows the concurrent reading of the locked object but prevents updates or deletion of the object.
- Exclusive Lock - prevents concurrent access of the locked object at any isolation level other than dirty read.
- Intent Lock - reflect the possible modification of an object of higher granularity. For example, during the modification of a page, RSAM creates an intent lock on the table in which the page is contained.
- Update Lock - allows proper queueing of lock requests and a greater level of concurrency. An attempt to secure an exclusive lock fails if a shared lock already exists on the requested object. In this case, an update lock is granted and the requesting process is blocked until the shared lock is released.

	Requested Lock						
Existing Lock	None	IS	S	U	IX	SIX	X
None	None	IS	S	U	IX	SIX	X
IS	IS	IS	S	U	IX	SIX	X
S	S	S	S	U	SIX	SIX	X
U	U	U	U	U	X	X	X
IX	IX	IX	SIX	X	IX	SIX	X
SIX	SIX	SIX	SIX	X	SIX	SIX	X
X	X	X	X	X	X	X	X

Table 4.1. Lock Upgrade Matrix

- Invisible Lock - allows a process with a higher sensitivity label to lock an object with a lower sensitivity label. This type of lock does not prevent a process at a lower sensitivity label from querying or even modifying such an object. Instead, the process that originated the lock (the process with the higher sensitivity label) is notified of any changes in the object in case that process wants to roll back to preserve the semantics of a read operation.

In addition, there are three hybrid lock types: shared intent (SI), exclusive intent (IX), and shared exclusive intent (SIX). Shared intent and exclusive intent arise when multiple components of a larger object are locked in either shared or exclusive mode. Shared exclusive intent locks are created when intent locks are required due to both shared locks and exclusive locks. Locks can also be upgraded during the life of a transaction as depicted in Table 4.1. This matrix represents the *potential* upgrade for a lock. The actual resulting lock depends on what lock(s) are currently held on the object and what the wait mode (the wait mode is discussed in the following paragraphs) is. The potential resulting lock is shown in the table at the intersection of the existing lock and the requested lock.

A lock can be placed on a database, table, row or page. The request to create a lock on an object is serviced by RSAM in a similar manner regardless of the type of object. A lock request is checked against existing locks held by the requesting process. If a duplication is detected no new lock is created. If a less restrictive lock is held on the same object by the requesting process, the existing lock is upgraded in accordance with the lock upgrade matrix (see Table 4.1).

If the requesting process does not hold a lock on the object, a new lock must be allocated from the Lock Table. If no locks are available on the free list, the request is denied. If a free lock is found, it is assigned to the data object and merged with the lock monitoring data structure in the global shared memory Lock Table.

If another process holds an incompatible lock, the request must be queued. Request queueing is dependent on the level of waiting that the requesting process is willing to accept. A *wait* level of waiting indicates that the process is willing to wait indefinately for the resource. This could result in a permanent deadlock. A *nowait* level of waiting will return control to the requesting process immediately if the lock request fails or after the lock is acquired on a successful lock attempt. A *wait(n)* level of waiting will cause the requesting process to sleep the amount of time specified. If a lock still cannot be acquired for the desired resource after the specified time period, a failure is returned.

A lock is released when the resource is no longer needed or when the locking process terminates. When a process terminates, all locks held by the process, as identified in the User Process Table in global shared memory, are removed.

4.4.8.2 Data Structure Integrity

INFORMIX-OnLine/Secure uses latches to guarantee exclusive access to global shared memory data structures to a single RSAM instance. Latching serves a similar function for global shared memory that locks serve for disk structures. Latches are implemented via OS semaphores (see page 11, "Informix in the Operating System Environment" for a complete discussion of OS services employed by INFORMIX-OnLine/Secure).

Latches provide a low-level exclusive global shared memory access control service on top of which the more sophisticated locking mechanism is built. For example, RSAM will latch the Lock Table entries or User Process Table entries as required to create, upgrade, or remove a lock. A latch is a structure that is part of the global shared memory data structure that can be latched and is in one of three states: FREE, BUSY, or BUSYWAIT. A BUSY latch always contains the address of a User Process Table entry while a FREE latch contains a NULL address.

A BUSYWAIT in indicated in one of two ways depending on the OS. If the OS implements *queued latches* (more than one process waiting on a latch), the presence of a non-empty queue on a BUSY latch indicates the latch is in the BUSYWAIT state. If the OS does not implement queued latches, a special flag is set in the latch structure that indicates a BUSYWAIT. This flag is needed since the test-and-set operation performed on the latch can only sense either BUSY or WAIT.

4.5 Session Startup

This section focusses primarily on the communication mechanism between the SQL Engine and the RSAM for each user session at startup for the B1/EA configuration. This communication mechanism is not used by the C2 and B1/EP configurations as they execute in the same UNIX process as the RSAM. Consequently, they communicate with the RSAM using UNIX pipes.. All users, including untrusted users, DBSAs and DBSSOs utilize the same communication protocol. However, each user type is authorized a different set of commands which they can execute.

When a user initiates a new session with INFORMIX-OnLine/Secure the SQL Engine creates a small shared memory segment (termed session shared memory) for communication with the RSAM process. The SQL Engine process ID is used as the key in the creation of the session shared memory segment. This key is provided to the RSAM process as an argument to the *exec* system call. The SQL Engine and the RSAM process both attach to the session shared memory segment using the *shmat* system call. The session shared memory segment has the same sensitivity label as the SQL Engine. The OS DAC permissions on the session shared memory segment are set such that only the RSAM process and the SQL Engine can read from or write to the little shared memory. The RDBMS Kernel creates two pipes, one for the SQL Engine and one for the RSAM process, to be used for read/write coordination.

Communication between the SQL Engine and the RSAM process begins when the SQL Engine sends a bit through the pipe to the RSAM process which indicates that the SQL Engine has put a message in the session shared memory segment. The RSAM process picks up the message, performs the required activity, and places the results back in the session shared memory segment. The RSAM process sends a bit through the pipe and the SQL Engine picks up the returned message from the session shared memory. If the size of the message from the SQL Engine is larger than the session shared memory segment size, the SQL Engine writes the maximum possible amount of the message and sets the "more" flag, which is the first byte of the session shared memory segment. The RSAM process recognizes that the "more" flag is set, copies the request to its internal memory, and releases control of the session shared memory segment back to the SQL Engine. The SQL Engine can now send the remainder of the message. Conversely, if the size of the resulting message from the RSAM process is larger than the session shared memory segment size, the RSAM process sets the "more" flag and the communication continues similar to that described above.

When all communication is completed for this session, the SQL Engine deletes the session shared memory segment and pipes. The RSAM process receives an error message due to the deletion of the pipes and terminates.

The messages passed between the SQL Engine and the RSAM process are grouped according to request type. Specifically, data manipulation, data access, data definition, data integrity, and supporting utility requests. The messages are well defined with message IDs, request codes, arguments for data types and range limits, and legality checks.

4.6 Support and Transient Processes

In INFORMIX-OnLine/Secure the DBSA must be able to perform routine maintenance . This involves such things as tuning the configuration of the RDBMS, extracting statistical information, and performing maintenance of internal RDBMS structures. To perform such maintenance tasks, the DBSA invokes INFORMIX-OnLine/Secure processes (called *support processes*) which are untrusted with respect to the OS and allow the DBSA to monitor and modify internal RDBMS data structures in a restricted manner.

The DBSA initiates a support process, by invoking its associated *transient process* which is trusted to bypass the OS MAC mechanism, making them trusted processes in the context of the OS. The transient process invoked determines whether the invoker is the DBSA and is logged in at the appropriate sensitivity label to execute the requested function. After making the access checks, the transient process establishes communication with the invoking process and then upgrades its sensitivity label to **Datahi+IX_DATA** and changes its group ID to **ix_data** so it can access information stored in global shared memory and raw devices. The transient process then spawns the associated single-level support process which performs the requested function. The support process passes information back to the transient process, which in turn passes it back to the requesting DBSA. Figure 4.10 illustrates the information flow when a support process is invoked through a transient process.

Although support processes are untrusted with respect to the OS security mechanisms, they remain trusted in the context of the composite TCB because they have access to initialization data during INFORMIX-OnLine/Secure start-up and multi-level database data on devices. The DBSA is the only user who can invoke a support process through its associated transient process. The primary function of transient processes is to spawn support processes. They are the only administrative utilities which are trusted to bypass the operating system security mechanisms and allow all support processes to execute at a single OS sensitivity label (i.e.,**Datahi+IX_DATA**) as a member of the DAC group **ix_data**.

The DBSA is required to run support processes that monitor system activity at **Datahi+IX_DBSA**. Support processes that alter the DBMS configuration or tune the database in any manner must be run at **Datalo+IX_DBSA**. Table 4.2 details the INFORMIX-OnLine/Secure support processes, their corresponding transient processes and the level from which the transients must be invoked. Following is a discussion of each of the support processes in INFORMIX-OnLine/Secure.

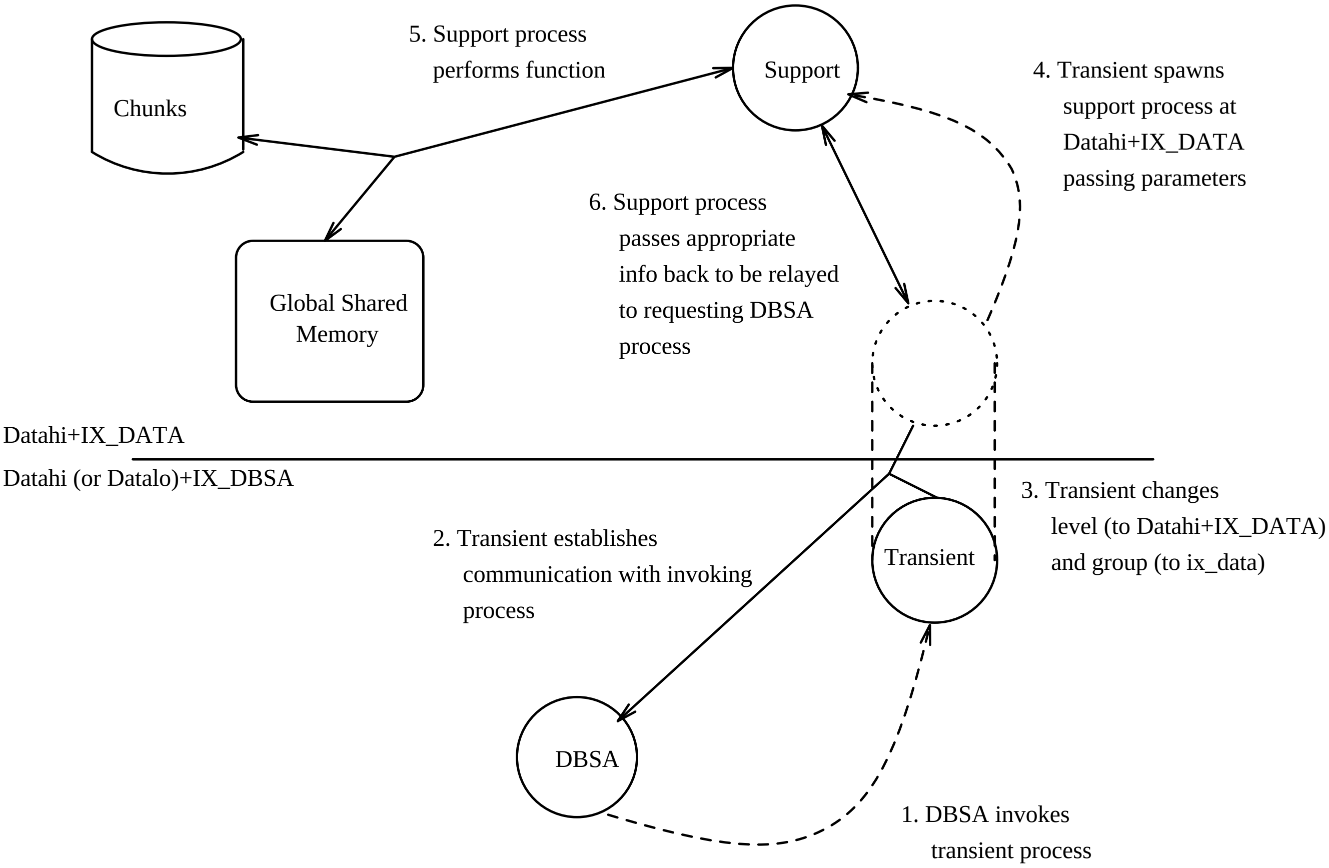

5. Support process performs function
Support
4. Transient spawns support process at Datahi+IX_DATA passing parameters
Chunks
6. Support process passes appropriate info back to be relayed to requesting DBSA process
Global Shared Memory
Datahi+IX_DATA
Datahi (or Datalo)+IX_DBSA
3. Transient changes level (to Datahi+IX_DATA) and group (to ix_data)
2. Transient establishes communication with invoking process
Transient
DBSA
1. DBSA invokes transient process

Figure 4.10. Launching Support Processes

Transient Process	Support Process	Required DBSA Sensitivity Label
tbcheck	*stbcheck*	**Datahi+IX_DBSA**
tbunload	*stbunload*	**Datahi+IX_DBSA**
tbload	*stbload*	**Datahi+IX_DBSA**
tbstat	*stbstat*	**Datahi+IX_DBSA**
tbinit	*stbinit*	**Datalo+IX_DBSA**
tblog	*stblog*	**Datahi+IX_DBSA**
tbmirror	*stbmirror*	**Datalo+IX_DBSA**
tbmode	*stbmode*	**Datalo+IX_DBSA**
tbparams	*stbparams*	**Datalo+IX_DBSA**
tbspaces	*stbspaces*	**Datalo+IX_DBSA**
tbtape	*stbtape*	**Datalo+IX_DBSA**

Table 4.2. Support Processes

4.6.1 Stbcheck

The *stbcheck* support process retrieves statistical information on indexes and RDBMS Kernel data structures. It retrieves information to ensure that no tblspaces/chunks overlap, that all tblspaces/chunks can be opened and reports the size of a given tblspace/chunk. For each system catalog table, among other things *stbcheck* validates the table against the partition description, reads all syscolumns and validates the number of columns and the rowsizes. For indexes, *stbcheck* lists the indexes associated with a particular table along with the row ids to which the index points.

4.6.2 Stbunload

Stbunload is invoked to use write data in a database or table to secondary storage in binary format. Only the DBSA has the privilege to invoke this program and can only use it when the system is in either the on-line or quiescent state. Data being transferred to secondary storage must be formatted in a specific order to be subsequently restored.

4.6.2.1 Tape Format

There are two possible formats for the tape, one for unloaded databases and the other for unloaded tables. The backup tape format consists of a number of regions. The regions are: the label mapping region, the BLOBspace region, and the data page region. The first two regions are common to both formats. The data page region differs between tapes and databases. Each region is delineated by a *separator* page which contains general formatting information (e.g., length of entries, address of initial entry) for that region.

The first thing to be written to tape is a Tape Header. The Tape Header page tells whether the tape contains an unloaded table or an unloaded database, thus driving the expected format of the tape. The Tape Header page also serves as the initial separator page for the label mapping region of the tape. The label mapping region contains the external representation (ASCII) of all sensitivity labels that are in the database or table and maps them to their internal operating system representation. The information in this region is used to map labeled information on the tape to the target operating system labeling scheme.

Next on the tape is the BLOBspace region. The BLOBspace region maintains all the BLOBspaces associated with a database or table being transferred. The Data Page Region follows the BLOBspace region. This region stores all the user information and table information associated with the table or database being unloaded.

Table Format For the table tape, the data page region has three subregions: the system catalog subregion, the bndlspace subregion, and the tblspace data subregion. Each subregion comprises one or more table tblspace descriptors and tblspace data pages. The table tblspace descriptor provides information about the tblspace to which it is associated.

The system catalog subregion contains all the information extracted from the system catalog tables which is needed to define the table being unloaded. Following the system catalog subregion is the bndlspace subregion. This subregion contains bndlspace information associated with the table in the database. The last subregion is the tblspace data subregion which stores user data associated with the tblspaces comprising the bundles.

Database Tape For a database tape, there are only two subregions: the system catalog subregion and the tblspace data subregion. As with tables, each subregion comprises one or more table tblspace descriptor/tblspace pairs. The system catalog subregion contains all the system catalog tables for a database instead of the information for just one table. As with the table format, the tblspace subregion contains the tblspaces of all the tables in the database. There is no need for a special bndlspace subregion because all tblspaces (including the bndlspaces) are unloaded to tape therefore eliminating the need to extract bndlspace information separately.

4.6.2.2 Execution

Before exporting a database or table, *stbunload* ensures that global shared memory is present and that all chunks are accessible. In addition, a tape must be available for use and assigned to the RDBMS with a label. It then allocates a buffer which it uses as an intermediate storage area for pages migrating from disk to tape.

To begin unloading a table, *stbunload* opens the database in which the table resides and accesses the **systables** system catalog table to locate the table. The row identifying the table is locked. In preparing the target tape for the transfer, *stbunload* then opens the tape device and writes a tape header which contains such information as the timestamp, the database/table flag, a tape counter and the tape size.

The Label Map region is then unloaded. *Stbunload* steps through the bndlspace of the table to be unloaded and finds all the sensitivity labels that are used in that table and writes the ASCII representation along with its internal operating system representation to tape so that they can be appropriately mapped back to the operating system sensitivity labels when the information is reloaded. Although the ASCII represenation is written it may not be trusted. This is because truncation may have occured with long sensitivity labels. The operating system administrator should be consulted to verify the ASCII representation.

Stbunload then determines if there are BLOBspaces associated with the tables to be unloaded. Then, the table is unloaded. First the bndlspace information is unloaded into the bndlspace subregion. Then tblspace data is unloaded. To unload the tblspace data, *stbunload* opens the tblspace and sets a pointer to the first extent for the table. It then unloads the tblspace description into the table tblspace descriptor. Afterwards, for each extent, it writes all the pages in the extent to the tblspace data subregion.

Once a tblspace has been unloaded to the buffer, if it has tblspace BLOBS associated with it, these BLOBS are stored immediately after the tblspace. After the last tblspace has been unloaded, a tape trailer is written.

In unloading a database, much of the processing is the same. The label mapping region and the BLOB region processing remain the same except that all the bndlspaces and **syscolumns** are searched for sensitivity labels and BLOBspace definitions. In processing the data page region, instead of picking out information associated with a specific table, the whole system catalog, all the bndlspaces, and all of the tblspace data are unloaded to tape.

4.6.3 Stbload

Stbload is invoked to load data from a device which has been been previously saved using *stbunload.* If the tape contains pages for an entire database, *stbload* creates a database; if the tape contains pages from a single table, *stbload* creates a table in the specified database. Only the DBSA can invoke this program;

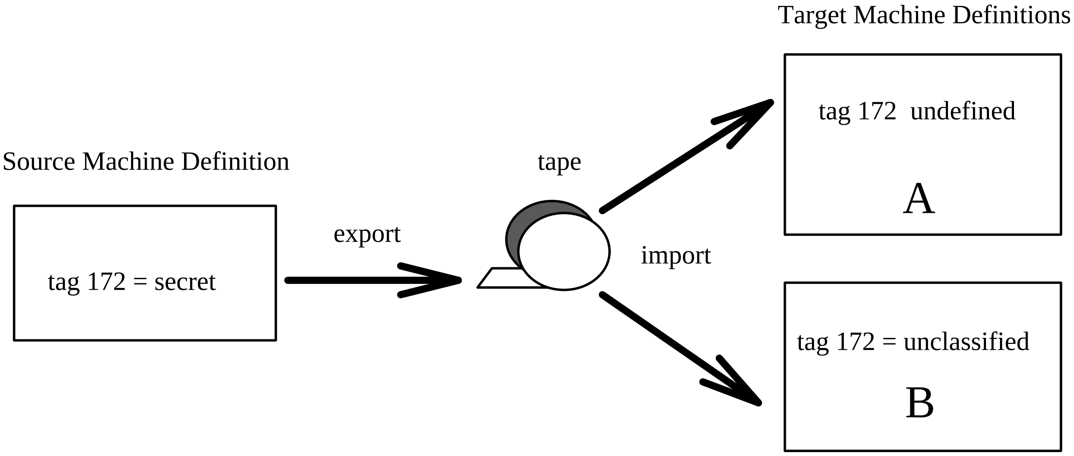


Figure 4.11. Data Importation and Labeling Discrepancies

however, before invoking *stbload*, the DBSA must ensure that the ASCII labels and the internal sensitivity label representation of the target operating system are compatible with the ASCII and internal sensitivity label representation of the source operating system stored on the tape. This section discusses what the DBSA, DBSSO and Operating System Administrator (OSA) must do to ensure this.

4.6.3.1 Preparation

Objects being imported using *stbload* have the same sensitivity labels as when they were initially placed on the tape. If the sensitivity label definition was changed or removed between the time when the data was exported and the time when the DBSA imports it, the DBSA runs the risk of importing objects for which there are no sensitivity labels defined in the system or for which the sensitivity label means something else. Figure 4.11 demonstrates the potential problems associated with incorrectly importing information which is not properly mapped into the sensitivity label definition. In this example, 172 (the internal OS representation) is associated with the secret sensitivity label when the information was unloaded. If the DBSA imports information to a system which does not define 172 with any sensitivity label (situation A), the objects labeled with the undefined sensitivity label will be inaccessible to everyone, including the DBSA and DBSSO. If the DBSA imports data to a system which contains labels that are defined differently (situation B) in the target operating system, the objects labeled with the different sensitivity label inherit the new string representation of that sensitivity label (unclassified) and its meaning. The objects with the newly defined sensitivity label will need to be relabeled by the DBSSO.

To prevent these types of situations, INFORMIX-OnLine/Secure provides a mechanism to map sensitivity label representations of imported data to the appropriate internal OS numeric representations maintained by the target operating system. To implement this mechanism, the DBSSO and OSA must cooperate to determine the appropriate mapping.

The DBSSO must create a sensitivity label map file called **label.map**. The sensitivity labels (and their numeric representation) on the target operating system and the tape must be known. Before the tape is unloaded, the DBSA must request a copy of the labels file from the OS. That labels file is stored with the tape. When the DBSA is ready to load the tape, the OSA from the target machine provides the labels file from the target machine. The DBSA then takes the two lables files and creates the label.map file. This mapping preserves the label representation associated with the data on the tape. *Stbload* provides an option to determine the sensitivity labels and the numeric representation from the tape; the OSA must provide the label representations for the target operating system. After receiving the two lists, the DBSSO compares the lists and populates the **label.map** file with each line comprising numeric entries: entry one from the tape and entry two from the operating system which will replace entry one when the data is imported. After **label.map** has been created, the DBSA can successfully run *stbload*, which translates the sensitivity label on the media into valid labels within the target operating system.

4.6.3.2 Execution

Before loading a database or table, *stbload* makes sure global shared memory is present, opens the tape device, and reads the tape header to determine if a database or table is being loaded. *Stbload* then looks for the label map file; if the file is not present, *stbload* terminates. If the label map file is present, *stbload* checks if label mapping is required. If it is required, then the file must contain a sequence of numerical values pairs (internal operating system representation of sensitivity labels) detailing the mapping. *Stbload* validates the label map file to make sure that there is a a unique mapping for each label on the tape. If not, *stbload* terminates with an error.

If the label map file is valid, *stbload* then searches forward on the tape for the BLOBspace separator page. After finding this page, for each BLOBspace that is read from this region, *stbload* queries the DBSA as to where this BLOB should be placed. After all BLOBs have been moved to disk, *stbload* looks for the data page separator where it extracts the system catalog information and inserts a row into the target database's **systables** and locks it.

Once done with BLOBspaces mapping setup, the next region separator is found. When loading a table, *stbload* extracts system catalog information from the system catalogs regions, inserts a row in the target **systables** for the table being loaded, and locks this row. In doing this, a new tabid is assigned to the table.

For each table being loaded, the bndlspace is loaded first, then for each row in the bndlspace *stbload* creates a new tblspace and transfers the data in that tblspace to disk, inserting the new tabid into the appropriate field in the bndlspace row. After the tblspace is loaded, any BLOBs residing in the tblspace are loaded. The BLOB location in the corresponding column tuple-blob structure is updated as required.

If a database is being loaded, *stbload* creates a new database tblspace in the root dbspace by default, unless the name of the dbspace where the database is to loaded is specified. The process of loading is similar, but the table loading process is repeated for each table in the database, starting with the system catalogs and following with the user tables. Additional processing happens at the beginning with the creation of the database and the insertion of a new row in the database tblspace for the new database.

4.6.4 Stbstat

Stbstat is a utility to monitor the system. Each option displays RDBMS statistical information about: chunk usage, physical and logical log buffers, user profiles, latches, global and shared memory.

4.6.5 Stbinit

Stbinit is used to start INFORMIX-OnLine/Secure or to initialize the root dbspace. During initialization, *stbinit* initializes the root dbspace (destroying all databases in the process) and starts INFORMIX-OnLine/Secure in the on-line state. *Stbinit* can also be invoked to bring INFORMIX-OnLine/Secure into the quiescent state. A detailed discussion of what *stbinit* does to initialize INFORMIX-OnLine/Secure is found on page 67, "Daemons".

4.6.6 Stblog

Stblog is a utility used to display the contents of one or more logical logs. It provides options to retrieve activities associated with a specific user, table or transaction number. When executed, it retrieves information such as the log record address, the transaction number associated with the activity, the logical log number and the activity type.

4.6.7 Stbmirror

Stbmirror is used to monitor and maintain the mirroring status of dbspaces and BLOBspaces. Through this utility, a DBSA can invoke mirroring on a specific dbspace or BLOBspace, request the path of a primary or mirrored space, list the primary chunks in a dbspace or BLOBspace and remove mirroring. *Stbmirror* invokes/revokes mirroring by setting appropriate flags in global shared memory.

4.6.8 Stbmode

Stbmode is invoked by the DBSA to change the operating state of INFORMIX-OnLine/Secure or force certain conditions in the system. *Stbmode* performs state changes by altering the mode parameter in global shared memory and waking up the master daemon to alter other daemon states if necessary (e.g., spawn them if bringing the system to the online state, kill them if bringing the system to the offline state).

In addition to changing states, *stbmode* provides the administrator with the following functions:

- force a checkpoint,
- force the switch to a new logical log,
- kill a server process, and
- toggle the global shared memory residency flag which allows global shared memory to be swapped to disk if necessary.

4.6.9 Stbparams

The *stbparams* utility is used by the DBSA to add or drop a logical log, or change the size or location of the physical log. To drop a logical log, the DBMS must be in the quiescent state.

4.6.10 Stbspaces

Stbspaces is used to create or drop a dbspace or BLOBspace, add a chunk to a dbspace or BLOBspace, or change the status of a chunk. See page 33, "Dbspace Management" or page 37, "BLOBspace Management" for a detailed discussion of dbspace and BLOBspace management.

4.6.11 Stbtape

Stbtape invokes and manages system archiving routines, logical logging status, and backups. An archive is a copy of the entire INFORMIX-OnLine/Secure system that reflects the status of the data at one point in time. Archiving is done in essentially two ways: *data* is archived to capture the contents of the RDBMS at regular intervals, and *transactions* are archived to capture transaction changes that are made to the system between points when the data is archived.

INFORMIX-OnLine/Secure supports three archiving strategies to back up the data in the system: level 0, level 1 and level 2. Level 0 is a complete archive. It backs up all the data in the system. The contents of all pages of all tables of all databases are written to tape. Level 1 backs up data that has changed since the last level 0 backup. A Level 2 archive backs up data that has changed since the last level 0 or level 1 archive, whichever is more recent.

Logical log backup capture changes that are made to the system between data backups by recording the transactions that were committed since the last data archive. *Stbtape* provides specific options to:

- back up each logical log to tape as it becomes full,
- back up all logical logs at a single point,
- archive/restore (level 0, 1, 2) data in the system,
- change the logging status to buffered/unbuffered for a particular database, and
- terminate logging for a particular database.

4.7 Daemons

Daemon processes are used by INFORMIX-OnLine/Secure to perform periodic housekeeping chores and to perform startup and shutdown services. One daemon, the *master daemon*, is started when INFORMIX-OnLine/Secure is started and remains running until the system is shutdown. The master daemon may fork off a *cleanup daemon* that cleans up after dead user processes (a front end process that exited without notifying its RSAM process) and *page cleaner daemons* that flush global shared memory buffers to disk. This section describes the functions performed by the master daemon, the cleanup daemon, and the page cleaner daemon.

4.7.1 Master Daemon

The master daemon is created during the initialization of INFORMIX-OnLine/Secure and performs a number of initialization, service, and shutdown functions. The master daemon is responsible for creating and managing the global shared memory, for moving the system between states, and for managing all the other daemons. The following sections give details about the startup of INFORMIX-OnLine/Secure and how the master daemon is created, give details about INFORMIX-OnLine/Secure shutdown, and describes the periodic functions performed in the master daemon service cycle.

4.7.1.1 System Startup

INFORMIX-OnLine/Secure goes through a number of steps during startup. This section first describes the steps followed when starting INFORMIX-OnLine/Secure for the first time. Then the differences in steps when restarting INFORMIX-OnLine/Secure are given followed by descriptions of starting up by restoring the system from tape or from a INFORMIX-OnLine/Secure abort.

INFORMIX-OnLine/Secure is started the first time by issuing the *tbinit* command with the -i option. The transient process *stbinit* is started. After processing the command line options *stbinit* reads the configuration file set up by the DBSA and stores the configuration information in its process memory. The configuration file contains the following information:

- the root dbspace name, pathname, and size
- the disk mirroring pathname and size
- the physical log dbspace and size
- the number and size of logical logs
- the system log pathname
- the console message pathname
- the system archive tape device pathname and tape block size
- the logical log tape backup configuration
- the INFORMIX-OnLine/Secure server name and number
- the global shared memory forced residency flag
- the number of concurrent INFORMIX-OnLine/Secure users
- the number of locks, buffers, open tblspaces, chunks, dbspaces, page cleaners, and other items used to generate the size of global shared memory
- the global shared memory base address
- the checkpoint interval
- the page size

Sanity checks are then performed by *stbinit* on the values obtained from the configuration file.

Stbinit next spawns the master daemon and sleeps. When the master daemon is finished with initialization and there have been no errors, a signal is sent to the sleeping *stbinit* and *stbinit* will exit. If there have been errors *stbinit* will wake up and terminate with an error.

The first step the master daemon performs is to create and attach to global shared memory. The base address of the global shared memory was obtained from the configuration file. The size of the global shared memory allocated is calculated from the size of the header and the sizes of table entries indicated in the configuration file.

The global shared memory is initialized next (see page 40, "Global Shared Memory" for the description of its contents). Entries for the master daemon, the cleanup daemon, and page cleaner daemons (if any) are placed in the User Process Table. The rest of the User Process Table entries are placed on the free list. All the Lock Table entries are put on the free list. All the Buffer Table entries are evenly distributed on the

LRU lists and all are marked empty. All the Tblspace Table entries are placed on the free list. All the Page Cleaner Table entries (in the Flush Control Structure, see page 40, "Global Shared Memory") are initialized.

The master daemon then goes on to initialize some local data structures and the root dbspace information. The root chunk information from the configuration file is verified first. If mirroring is enabled then the mirror chunk information if verified also. Root dbspace reserved page structures are set up to accommodate the page structure and each set of reserved pages of the root chunk of the root dbspace are initialized. The database tblspace, tblspace-tblspace, physical log space, and logical log space are also initialized.

At this point, if there have been no errors, the master daemon sends a signal to *stbinit* so that it can exit. The master daemon then writes all the initialized root chunk pages to disk. A checkpoint is performed at this time and the master daemon moves INFORMIX-OnLine/Secure to the quiescent state. The configuration parameters copied from the configuration file are then written to the second page of the root chunk of the root dbspace.

The master daemon may move INFORMIX-OnLine/Secure into the on-line state if that option was indicated, otherwise it will remain in the quiescent state. At this point the one time initialization is done and the current environment is saved (called the "save point") for use in the case an abort condition is detected. A typical abort condition is when a front end process exits while holding a latch.

If INFORMIX-OnLine/Secure is being restarted the steps followed are similar to the steps outlined above. The command line options will indicate whether to start and go into the on-line state or go into the quiescent state. The root dbspace initialization involves reading in and verifying the information stored on the root chunk of the root dbspace rather than initializing the pages and writing them to disk as described above. The size needed for global shared memory is calculated using the reserved pages in the root chunk of the root dbspace. The master daemon will also perform fast crash recovery at this point if there has not been a media failure (e.g., there was a power failure or the OS failed). If the physical log has entries the database has been modified since the last checkpoint and fast recovery automatically occurs. The configuration parameters are only written to disk if they have been changed.

INFORMIX-OnLine/Secure can be started after a complete restore from an archive. To do this the utility *stbtape* is used with the -r option. After completing the restore from tape, *stbtape* executes *stbinit* and INFORMIX-OnLine/Secure comes up in quiescent mode after the restore. INFORMIX-OnLine/Secure can also automatically restart from the Abort state. The master daemon begins executing from the "save point" and starts the initialization routine from the point where global shared memory is allocated and attached (after releasing the existing global shared memory).

4.7.1.2 Shutdown

The DBSA has several options for moving INFORMIX-OnLine/Secure from the on-line state to either the quiescent state or the off-line state. The support process *stbmode* is used. INFORMIX-OnLine/Secure can be shutdown in a graceful manner with the -s option. The master daemon first ensures that all dead user processes are taken care of up by waking up the cleanup daemon. The master daemon then waits for all active user processes to exit. The master daemon then moves to the quiescent state, does a final check for dead user processes and has the cleanup daemon exit after it is through processing. The master daemon then performs a checkpoint. At this point the DBSA can bring the system to the off-line state or perform maintenance and return to the on-line state. The global shared memory is not deallocated with this option.

If the -u option is used with *stbmode* then all user processes are killed immediately and INFORMIX-

OnLine/Secure is moved to the quiescent state. The *stbmode* places a *kill* command in the user structure and sends a signal to each of the user processes to process the *kill* command. If the -k option is used with *stbmode* all user processes, the page cleaner daemons, and the cleanup daemon are killed; the global shared memory is deallocated; and INFORMIX-OnLine/Secure is moved to the off-line state.

4.7.1.3 Service Cycle

Once initialization is done the master daemon starts a service cycle. The master daemon goes to sleep and sets an alarm to wake up every second. The master daemon has an alarm handler when it wakes up. The handler first checks to see if an abort has been requested (see below). If there has not been an abort then the following steps are performed:

- if INFORMIX-OnLine/Secure is in the shutdown state then the shutdown processing described above is performed
- dead user processes are cleaned up if they do not have any unfinished transactions
- all other daemons whose alarms have expired are woken up
- locked buffers that a page cleaner could not flush are written to disk
- change to a different logical log if requested by the DBSA
- start a checkpoint if requested by another process

Every 15 seconds INFORMIX-OnLine/Secure system time kept in global shared memory is updated with the OS time. Every 30 seconds the master daemon creates (if necessary) or wakes up the cleanup daemon to handle dead user processes that may need transactions rolled back and flushes table headers to disk. Every 5 minutes the master daemon starts a checkpoint.

If INFORMIX-OnLine/Secure is in the abort state when the master daemon wakes up then INFORMIX-OnLine/Secure starts execution from the "save point." All user processes, the page cleaner daemons, and the cleanup daemon are killed. Semaphores are removed and global shared memory is deallocated. If the reboot flag is on then the master daemon starts initialization by creating global shared memory.

4.7.2 Cleanup Daemon

The cleanup daemon is spawned by the master daemon the first time any user process dies abnormally leaving a transaction that needs to be rolled back. The cleanup daemon goes through a cycle of sleep, wake-up, and cleanup after dead user processes. While awake, the cleanup daemon goes through the User Process Table, finds those user entries that need a rollback service, closes off all open tblspaces for those dead user processes, and performs a rollback on the current transaction. Any locks held by the dead user are freed, and then the User Process Table entry is released. The cleanup daemon makes sure that no user entry needs cleanup before it goes to sleep. When the INFORMIX-OnLine/Secure goes into the quiescent state, the cleanup daemon detaches from global shared memory and exits.

4.7.3 Page Cleaner Daemon

Page cleaner daemons (page cleaners) are used to flush dirty page buffers to disk so that the RSAM processes will have available buffers when needed. There can be up to 32 page cleaners configured in INFORMIX-

OnLine/Secure by the DBSA. If there are no page cleaners configured or if all the page cleaners are busy, then the master daemon performs the page cleaner functions.

Page cleaners execute five basic functions:

- LRU cleanup
- physical chunk flushing
- near flushing
- idle service
- shutdown

The function that should be performed is placed in a Page Cleaner Table entry. When the page cleaner wakes up it retrieves the command and executes accordingly. The five functions are described below.

A page cleaner's normal function is to flush dirty pages from LRU lists to keep a number of clean buffers available for RSAM processes to use. The LRU lists are divided evenly among page cleaners. When a page cleaner wakes up it goes through its LRU lists and looks for the oldest buffers which are unlocked and dirty. Those buffers are written to disk one by one until the minimum number of allowed dirty buffers is reached. A page cleaner will not flush an LRU list if the number of dirty buffers is below the maximum. The minimum and maximum thresholds are based on the percent of dirty pages in LRU list.

Physical chunk flushing is performed when disk flushing activities require fast disk I/O. A buffer's entry in the Buffer Table contains information indicating the buffer's physical chunk. The page cleaners use this information to group sets of pages together and write the whole group to disk in one disk write. Near flushing takes advantage of pages that are physically near each other on the disk. Flushing pages near each other decreases the disk read/write head seek times. Page cleaners use a buffer's near list to find other pages to write.

A page cleaner's normal operation happens when its command indicates the page cleaner should perform the idle service (the default command). During idle service the page cleaner looks through its assigned LRU lists and cleans any that have exceeded a maximum number of dirty pages. After cleaning up all its LRU lists a page cleaner calculates how long it should sleep based on the amount of work it just performed, any time between one and 60 seconds. If a page cleaner receives the shutdown command then the page cleaner detaches from global shared memory and exits. This can occur when INFORMIX-OnLine/Secure is in the shutdown state or if the page cleaner timed out while flushing a chunk.

A page cleaner can be woken up before its calculated time by the master daemon for two reasons: when there are too many dirty buffers in the LRU lists (the LRU cleanup command is indicated to the page cleaner) and when a dirty buffer is found through the hash table (the near flushing command is indicated to the page cleaner).

4.8 Multiple RDBMS Instantiations

INFORMIX-OnLine/Secure can run in multiple instantiations on a single machine. Each instance of INFORMIX-OnLine/Secure has its own set of objects which it protects via DAC and MAC. Each instance is completely distinct and separate and does not interfere with any other instance. This separation is achieved through the use of distinct root dbspaces, shared memory, and a server number.

There are two environment variables of concern in INFORMIX-OnLine/Secure: $TBCONFIG and $INFORMIXDIR. The $INFORMIXDIR environment variable defines the pathname for the INFORMIX-OnLine/Secure system software. More than one INFORMIX-OnLine/Secure system can be in the same $INFORMIXDIR directory or in different directories. Each instantiation must have its own **tbconfig** file, however. INFORMIX-OnLine/Secure looks for the **tbconfig** file in **$INFORMIXDIR/etc** with the filename **tbconfig**. For multiple instantiations in the same $INFORMIXDIR, the environment variable $TBCONFIG is used to indicate the name of the **tbconfig** file for each instantiation. The **tbconfig** file identifies the server number that is a unique identifying number for each instantiation. This server number is used to set up global shared memory and subsequently attach to it. The **tbconfig** file also contains the pathname of the root dbspace which identifies all the other chunks defined for that instance of INFORMIX-OnLine/Secure.

In the B1 configurations, there is a single definition for **DataHi** and **DataLo**. The OS administrator assigns these values and they pertain to all instantiations on a single machine. There are also two groups on a single machine, **ix_dbsso** and **ix_dbsa**, and in the B1 configurations, two categories, **IX_DBSSO** and **IX_DBSA**. If a user is a member of these groups, and has the appropriate category in the B1 configurations, that user can perform the functions of the DBSSO and DBSA, respectively. Such users are privileged and trusted to administer any INFORMIX-OnLine/Secure instantiation on a single machine. There is no way to set up separate DBSSO/DBSA roles for a particular instantiation.

Thus, each instance of INFORMIX-OnLine/Secure protects its own objects via DAC. In the B1 configurations, using the services of the OS, each instance of INFORMIX-OnLine/Secure protects its own objects via MAC. All INFORMIX-OnLine/Secure instances store audit records in the same OS audit log. Each audit record contains the unique server number which distinguishes audit records from different instances. All users are still required to login to the OS and must be a member of the group **ix_users**, and in the B1 configurations have the category **IX_USERS**, prior to using any instance of INFORMIX-OnLine/Secure. Object reuse is performed by each INFORMIX-OnLine/Secure instance on its own chunks and global shared memory; each INFORMIX-OnLine/Secure only accesses chunks that were assigned to it. The system architecture requirement is still met since multiple instances are separated via a distinct root dbspace and set of chunks, shared memory, and server number, as previously described.

Chapter 5

DBMS Security Architecture

This chapter describes the protected resources (i.e., subjects and objects) in INFORMIX-OnLine/Secure. In addition the RDBMS protection mechanisms of DAC, MAC and Object Reuse are discussed. Finally, descriptions of auditing as well as administrative roles, including the DBSSO and SAFE, the DBSA, and the AAO are provided.

5.1 RDBMS Protected Resources

This section describes the portion of the TCB interface which INFORMIX-OnLine/Secure controls, the creation and deletion of RDBMS objects at that interface, and the security attributes of RDBMS subjects and objects.

5.1.1 INFORMIX-OnLine/Secure TCB Subset Interface

As illustrated in Figure 4.1 on page 22 and Figure 4.2 on page 23, there are two primary INFORMIX-OnLine/Secure architectures. In the C2 and B1/EP architectures, the SQL Engine and the RSAM execute in the same process. The SQL Engine in the B1/EA architecture is not part of the TCB as it executes in a separate process. The diagrams referred to above show the TCB boundary and explain all the different components. See page 21, "Architectural Overview" for more details on the separate components that comprise the TCB interface.

5.1.2 Subjects

Subjects are defined as UNIX processes running on behalf of users. All users access information in INFORMIX-OnLine/Secure via these subjects which inherit the security attributes of the users on whose behalf they are running. In INFORMIX-OnLine/Secure, users access objects through front-end processes which are considered to be the subjects for the RDBMS. INFORMIX-OnLine/Secure does not directly manage the introduction and removal of subjects. Instead, INFORMIX-OnLine/Secure relies on the services of the OS for process creation and manipulation. The binding of user's security attributes to subjects is also performed by the OS.

The initial sensitivity label associated with an RDBMS Kernel subject is equal to the sensitivity label of the user on whose behalf the subject is created. The user ID and group ID of the subject is obtained from the user's login information upon the subject's creation. The *fork* system call, generally followed by the *exec* system call, is used to create a process (subject). A process can either destroy itself with the *exit* system call or be terminated by its parent process with the *kill* system call.

NAMED OBJECTS	STORAGE OBJECTS
Databases	Databases
Tables	Tables
	Rows
Constraints	Constraints
Indexes	Indexes
View Definition	View Definition
Synonyms	Synonyms
Columns	

Table 5.1. INFORMIX-OnLine/Secure Objects

5.1.3 Objects

An object is an entity that contains information. Access to an object potentially implies access to the information that object represents or contains. This is done by the operating system. Unlike subjects, however, the set of objects in the RDBMS Kernel is disjoint from the set of objects in the operating system. RDBMS Kernel objects do not exist by themselves; they are stored in some operating system objects (such as files or raw devices) and these operating system objects are specifically dedicated for storage of RDBMS Kernel objects.

The RDBMS Kernel depends on the correct functionality of the device abstraction provided by the operating system to access the RDBMS Kernel objects stored there. The operating system must ensure that the contents of a device will not be altered by an operating system subject without going through the RDBMS Kernel interfaces. Each object is associated with a sensitivity label that is set to the sensitivity label of the creating subject when the object is created by th RDBMS Kernel (seepage 79, "Mandatory Access Control"). In addition, the owner field of an object is set to the user ID of the user who creates the object, although a user with **dba** privilege can create objects to be owned by others (see page 75, "Discretionary Access Control").

The following paragraph presents the named and storage objects protected by INFORMIX-OnLine/Secure. The named objects are objects which are protected by DAC when INFORMIX-OnLine/Secure is running in any configuration. The storage objects are objects protected by MAC in the B1 configurations. A brief description of each object and a description of the security attributes of each object (e.g., owner, access privileges, label) are given.

A database is a collection of related information, and has a creator, privileges, and labels. A table is an array of data contained in columns and rows. It also has an owner, privileges, and labels. A row in a table is one instance of information within that table; however, rows only have labels. A view definition is simply a logical table based on other tables and/or other views. Like a table view definitions have owners, privileges, and labels. Synonyms are alternative names for tables and view definitions. Synonyms have only owners and labels, no privileges. Constraints are the possible restricitions placed on data contained in a column. Like synonyms, constraints only have owners and labels. Finally, the object index is a quick access pointer to a table, with the security attributes of owner and labels.

5.2 RDBMS Protection Mechanisms

This section discusses the protection mechanisms provided by INFORMIX-OnLine/Secure. On page 75, "Discretionary Access Control", the discretionary access control mechanism (DAC) is discussed, page 82, "Object Reuse" describes how INFORMIX-OnLine/Secure controls the reuse of resources, and page 79, "Mandatory Access Control" discusses the mandatory access control mechanism available in the EA and EP configurations.

5.2.1 Discretionary Access Control

The INFORMIX-OnLine/Secure DAC mechanism has the ability to include or exclude access to RDBMS objects on a per user basis, and enables individuals to control other users' access to these objects. No user can access the information in a database unless that user has been authorized explicitly or by default to access it in accordance with the DAC policy[1]. The INFORMIX-OnLine/Secure DAC mechanism is completely separate from that of the OS, yet it *extends* the OS DAC policy by applying access attributes specific to RDBMS objects.

The RDBMS DAC policy protects information stored in databases up to the granularity of individual columns within given tables. The **syscolauth** and **systabauth** tables of the system catalog for a database can be thought of as an Access Control List (ACL) since they identify users and their access to objects within the database. DAC objects, also called named objects, correspond to databases, tables, views, synonyms, constraints, and indexes. DAC is accomplished via privileges[2] which users grant and revoke using SQL statements or the RSAM interface[3]. Privileges are granted to single users by name or to all users under the name of PUBLIC. There is no way to create groups or assign privileges to groups in INFORMIX-OnLine/Secure. The DBSSO has access to named objects without having the necessary INFORMIX-OnLine/Secure DAC privileges by being a member of the group ix_dbsso and, in the B1 configuration, having **IX_DBSSO** as part of the category set. In fact, one of the tasks of the DBSSO is to modify the privileges on named objects if the existing privileges pose a security threat.

Tables, views, synonyms, constraints, and indexes all have an *owner* which is the user who created the object. A database, on the other hand, has a *creator* which is awarded the **dba** privilege instead of ownership. Privilege is conceptually different from ownership. In order to facilitate transfer of administrative responsibility, the **dba** privilege for a database can be granted to and revoked from another user. The owner of a named object, on the other hand, remains fixed during its lifetime. In addition, one or more users can have the **dba** privilege for a given database, while only a single user can be the owner of a particular named object. A user that possesses the **dba** privilege is also called a DBA for that database. The next section discusses privileges in detail.

[1] In the B1 configurations, users must also be cleared for the information in accordance with the MAC policy as described on page 79, "Mandatory Access Control".

[2] The term "privilege" used here is not to be confused with operating system privileges. Instead, RDBMS privileges identify the type of access a user has to a database object.

[3] SQL statements are processed by the SQL Engine and passed in the appropriate format to the RSAM interface. Users can invoke the RSAM interface directly.

5.2.1.1 DAC Privileges

A user cannot access any information in a database unless that user has at least one of the following three *database level privileges*: **dba**, **resource**, **connect**. The **connect** privilege enables a user to access the database. This includes the ability to store retrieved information in temporary tables. Without the **connect** privilege, a user cannot have any access to the database. The **resource** privilege allows a user to create tables and indexes within a database, which is considered more privileged than simple accesses that retrieve or store information in the database. Finally, the **dba** privilege embodies the full administrative power in a database, including the ability to grant or revoke database level privileges to another user. The **dba** privilege, as with all other privileges, only applies within a particular database; it is not a system-wide privilege. A user who has the **dba** privilege on database d_1 may not have any privilege on another database, d_2. The three database level privileges are totally ordered. The possession of the **dba** privilege implies the possession of both the **connect** and the **resource** privileges. Possession of the **resource** privilege implies possession of the **connect** privilege.

Similarly, a user can access information in a table only if that user has at least one of the following *table level privileges*: **(a)lter**, **(d)elete**, **inde(x)**, **(i)nsert**, **(s)elect**, **(u)pdate**[4]. The **alter** privilege allows a user to change the relational schema of a table, as well as add or drop constraints on columns of the table. The **alter** privilege implies the **index** privilege. The **index** privilege allows a user to create an index on a table. The **delete** privilege allows a user to delete a row from the table while the **insert** privilege allows a user to insert a new row into a table. Neither the **delete** nor the **insert** privilege implies any other table level privilege.

Although there are no column level privileges per se, the **select** and **update** table level privileges can be granted on a certain column or columns. The **select** privilege allows a user to retrieve data from all or some of the columns in a table. The **update** privilege allows a user to change values in some or all of the columns of a table. The **update** privilege implies the **select** privilege.

DAC on views, synonyms, constraints, and indexes is controlled by database level and table level privileges. There are no specific privileges defined for these named objects.

There is a dependency between database level privileges and table level privileges. A table level privilege may not be exercised on a table if the user does not also have the required database level privilege on the database to which the table belongs. Conversely, even if a user has the necessary database level privilege, that user may not be able to access a particular table in the database if the user does not have the necessary table level privilege(s). The **alter** and **index** table level privileges depend on the user possessing the **resource** database level privilege. The **delete**, **insert**, **select**, and **update** table level privileges depend on the user possessing the **connect** database level privilege.

There are actually three different types of tables: permanent user tables, temporary user tables, and views. Permanent user tables have already been discussed. Temporary tables are created to complete operations, such as joins, and return information to the user. Upon termination of the user database session, any temporary tables are dropped. To create a view, a user must have the **connect** database level privilege as well as the **select** table level privilege on the base table columns to be used in the view. A user's table level privileges on a view can be equal to or less than that user's privileges on the base table; a user can never have more privileges on a view than that user has on the base table. Views can be created from other views, called *compound views*, however, there is no **update** privilege on compound views.

[4] The letter in parenthesis represents the INFORMIX-OnLine/Secure abbreviation for that privilege.

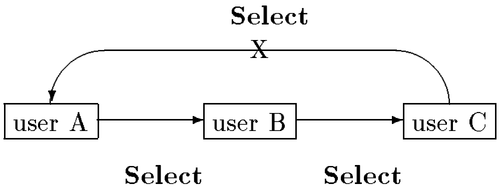


Figure 5.1. Prevention of grant cycles

5.2.1.2 Granting and Revoking DAC Privileges

The discretionary access of a user to a named object is represented by the privileges that the user possesses on a particular named object. While privileges are possessed by users, the DAC policy is applied to subjects which is determined by the privileges possessed by the user on whose behalf the subject is operating.

When a new named object is created, the *initial access* to it is established by default. If the named object is a database, the user who creates the database is given the **dba** privilege. If the named object is a table, the creating user becomes the owner and is given all table level privileges including the permission to grant and revoke these privileges to and from other users. If the named object is an view, synonym, constraint, or index, the creating user becomes the owner of the object but gets no specific privileges since none are defined for these types of named objects. No user other than the creator initially possesses any privileges to a named object with the exception of the DBA who has implicit privileges on tables as described below.

Changes in the discretionary access of a user to a named object can occur when a subject explicitly grants or revokes privileges to or from the user to the named object. Grants and revocations of privileges do not take effect immediately. They only affect any future access by the subject to the named object but do not affect the current access that the subject has already obtained.

A subject cannot grant or revoke a database level privilege (**dba**, **resource**, **connect**) to or from another user unless the subject possesses the **dba** privilege[5]. The granting/revoking of a database level privilege results in adding/deleting or modifying an entry in the **sysusers** table of the system catalog. If the user being granted the privilege currently has no database level privilege, an entry indicating the new privilege is inserted into the **sysusers** table of the system catalog for that user. If the user is being granted additional database level privilege, the entry in the **sysusers** table is modified to reflect the new privilege. If the privilege being revoked is **connect**, the entry for that user is deleted from the **sysusers** table. If the privilege being revoked is **dba** or **resource**, the entry in the **sysusers** table is modified so the user is left with the **connect** privilege.

[5] In the EA and EP configurations, the sensitivity label of the subject issuing the granting request must be equal to the sensitivity label of the database since a successful grant request will result in a modification of the **sysusers** table in the system catalog.

Grantee	Grantor	Tabid	Privilege
...	...	...	...
E	D	t_1	**Alter**
E	F	t_1	**Alter**

Figure 5.2. Duplicate table level privileges

Grantee	Grantor	Tabid	Privilege
...	...	...	...
E	D	t_1	**Alter**
E	F	t_1	**Alter**
H	E	t_1	**Alter**

Figure 5.3. Granting privilege on behalf of another user

A user, other than the table owner, possesses a table level privilege only if someone else previously granted that user the privilege. Each table level privilege can have a *grant* option associated with it which allows the user to give the privilege to another user. Table level privileges can be granted with or without the grant option. By convention, table level privileges with the grant option are represented with an upper case letter, e.g., **(A)lter**; table level privileges without the grant option are represented with a lower case letter, e.g., **(a)lter**. INFORMIX-OnLine/Secure prevents cycles of grants as shown in Figure 5.1. For example, user A grants user B the **(S)elect** privilege. User B then grants the same privilege with the grant option to user C. However, when user C tries to grant the **(S)elect** or **(s)elect** privilege to user A, the grant is denied because such a grant would constitute a cycle.

The granting of table level privileges results in the addition or modification of entries in the **systabauth** and **syscolauth** tables for each grantor/grantee pair. In other words, if user D grants **alter** privilege to user E on table t_1, and user F grants **alter** privilege to user E on table t_1, there will be two entries in the appropriate system catalog tables: one for user E/user D and one for user E/user F, as shown in Figure 5.2. If the grantee has no privileges on the table, an entry indicating the new privilege is inserted into **systabauth** for the grantor/grantee pair. If the privilege is explicitly granted for certain columns in a table, new entries are added into the **syscolauth** as well. If the grantee currently has certain privileges on the table and additional privileges are being granted by the same grantor, the existing entries for that grantor/grantee pair in the **systabauth** and **syscolauth** tables are modified appropriately. Neither the implied table level privileges of a user with the **dba** privilege on the database in which the table resides nor the implied privileges of the table owner is explicitly stored in the system catalog.

Possession of the **dba** privilege implies that the user has all privileges on all tables in the database without the grant option. A user with the **dba** privilege can grant table level privilege *on behalf of* another user as long as that user possesses the necessary privilege. For example, Figure 5.3 shows the **systabauth** table of the system catalog for database d_1 of which user G is the DBA. Several entries already exist in the **systabauth** table indicating that several users have access to the tables in database d_1, one of which is user E who has the **Alter** privilege on table t_1. As DBA, user G can grant user H the **Alter** or **alter** privilege on behalf of user E, but not on behalf of user G. In other words, after the grant operation performed by the DBA, the entry in **systabauth** has an entry for table t_1 with user H as the grantee and user E as the grantor.

A user can only revoke a table level privilege from another user if the revoking user originally granted the table level privilege. However, table level privileges can be indirectly revoked from a user through a chain of cascading effects. Consider the example in Figure 5.1. If user A subsequently revokes the **Select** privilege from user B, this privilege is also revoked from user C. However, in Figure 5.2, if user F revokes the **Alter** privilege from user E, user E will still have the privilege since it was also granted to user E by user D. Note that the grant option cannot be revoked separately. When a privilege is revoked, it is revoked regardless of whether it has a grant option associated with it. Users cannot revoke their own privileges.

When the **select** privilege is revoked on a table or view, any other views that are based on that table or view are automatically dropped. When any privilege other than **select** is lost on a table or view, that privilege is also revoked on any depending views. The revocation of privileges on a view does not affect the columns of the base table.

The revoking of table level privileges results in the deletion or modification of existing entries in the **systabauth** and/or **syscolauth** tables of the system catalog.

5.2.1.3 DAC Operations

The DAC mechanism mediates a rich set of RDBMS operations. Table 5.2 shows each operation and the database level and/or table level privileges required to perform the operation on a particular named object is listed.

A table can be opened in one of three modes: Input (read only), Output (write only), or Inout (read/write). The open mode is stored in the RSAM open table. When the table is opened, the only access check performed is to see if the user has *some* privilege on the table. If the user has no privileges on the table, the table open request will fail. If the open succeeds, the system catalog information is copied into RSAM local memory data structures and global shared memory as previously described. Subsequently, a DAC check is performed to ensure that the user has the appropriate privilege(s) when the actual operation on the table is attempted (see Table 5.2), using the information in RSAM local memory and global shared memory. Thus, if a privilege has been revoked for the user since the information was copied into memory, it won't effect the DAC check. A change in the privileges for the table will take effect after the table is closed. For example, to update a row, the table should be opened in the Inout mode; to select a row, the table need only be opened in the Input mode. Once the actual update_row or select_row operation is attempted, the DAC mechanism will check that the requesting process has the **update** (or **select**) privilege on the table that contains the row using the privilege information copied into memory.

5.2.2 Mandatory Access Control

INFORMIX-OnLine/Secure uses Mandatory Access Control (MAC) as a means of restricting access to objects by subjects based on the sensitivity of the information contained in the objects.[6] This control is achieved through the use of the sensitivity labels. Sensitivity labels are composed of a hierarchical classification and one or more non-hierarchical categories. The number of labels supported by INFORMIX-OnLine/Secure is dependent on the number of labels in the OS.

[6] Note that this section only applies to B1 configurations.

Object	Operation	Database Level Privileges	Table Level Privileges
Database	drop	**dba**	
	open	**connect**	
	close	**connect**	
	grant/revoke privilege	**dba**	
Table	create	**resource**	
	drop	1) table owner 2) **dba**	
	open	1) **connect** 2) **dba**	any table level privilege
	close	**connect**	
	get schema info	**connect**	
	alter schema	**resource**	**alter**
	rename	**resource**	**alter**
	insert row	**connect**	**insert**
	delete row	**connect**	**delete**
	select row	**connect**	**select**
	exclusive lock/select row	**connect**	**select**
	update row	**connect**	**update**
	create blob	**connect**	**insert**
	link blob	1) **connect**	**update**
		2) **connect**	**insert**
	open blob	**connect**	**select**
	grant privilege	**connect**	privilege with grant option
	revoke privilege	**connect**	grantor of privilege
View	create	**connect**	**select** on base table
	drop		view owner
Synonym	create	**connect**	
	drop		synonym owner
Constraint	add	**resource**	**alter**
	drop	1) **resource** 2)	**alter** constraint owner
Index	add	1) **resource** 2) **resource**	**alter** **index**
	drop	1) **resource** 2) index owner 3) **dba**	**alter**
	alter	**resource**	**index**

1. If a privilege is listed under database level and table level, *both* are required.
2. If more than one entry is listed for a particular operation, e.g., 1), 2), *only one* is required.

Table 5.2. DAC Operations and Required Privileges

The MAC policy is based on the dominance relation between sensitivity labels which is defined in the following manner. A label x is said to *dominate* label y if the following conditions hold:

- The classification of x is greater than or equal to the classification of y; and
- The category set of x is a superset of the category set of y.

Two labels are defined to be equal if both labels have the same hierarchical classification and set of categories.

Given this definition of label dominance, INFORMIX-OnLine/Secure implements the following security policy which is a more restrictive interpretation of the Bell-LaPadula security model:

- A subject cannot read an object unless the sensitivity label of the subject dominates the sensitivity label of the object.
- A subject cannot modify an object unless the sensitivity label of the subject equals the sensitivity label of the object.

Informix has defined several special sensitivity labels which are to be used when running the DBMS. **Syshi** is used to denote the highest access level at a particular site while **Syslo** denotes the lowest. In contrast, **Datahi** is used to identify the highest access level of data on the DBMS while **Datalo** identifies the lowest. It is important to note that **Syshi** and **Datahi** do not have to be equal; **Syshi** must dominate **Datahi**. Similarly, **Datalo** must dominate **Syslo**. A user session must dominate **Datalo** and be dominated by **Datahi** before the user is granted access to the DBMS.

Each TCB subset enforces MAC on its own set of storage objects. However, INFORMIX-OnLine/Secure extends the OS MAC policy to its objects by calling on the operating system to make MAC access decisions, passing these decisions back to INFORMIX-OnLine/Secure for enforcement. To do this INFORMIX-OnLine/Secure uses the operating system's internal and external representation of sensitivity labels. For each MAC decision made, INFORMIX-OnLine/Secure uses operating system service routines, which take sensitivity labels as input and provide dominance information as output. Upon reception of the output, INFORMIX-OnLine/Secure either grants or denies access accordingly. For a detailed description of each OS MAC policy, the Final Evaluation Report for each should be consulted [[5], [6], [7]]. For a detailed discussion of the services required from the OS platform to support INFORMIX-OnLine/Secure see page 12, "Required OS Services".

The sensitivity label of an OS subject is assigned by the operating system when a user forks a process. The sensitivity label of the DBMS subject is the same sensitivity label as that assigned by the operating system; DBMS subjects belong to a OS defined group **ix_users**. When a new MAC object is created, its sensitivity label is set to that of the subject that created it. There are seven MAC objects: databases, tables, rows, views, synonyms, constraints, and indexes. Before a subject may access an object, a MAC check must be performed against the sensitivity labels of the subject and object. In Table 5.3, the MAC permissions needed for all database capabilities are outlined. In this table, "dom" represents dominates, and "sl" represents sensitivity label. Following are descriptions of where the sensitivity labels are stored for MAC objects, and what MAC checks are made when each type of object is accessed. In addition to the object listed, locks have MAC restrictions as well. For a discussion of locks see page 57, "Lock Management".

Database sensitivity labels are stored in entries of the database tblspace. When the database is created, a new entry is made in the database tblspace and the assigned sensitivity label is the same sensitivity label as that of the creating subject. The sensitivity label represents the lowest classification of data that may be stored in the database.

The sensitivity label for a table is represented by the sensitivity label of the corresponding entry in the **systables** table of the system catalog. The table receives the sensitivity label of the subject creating it. The sensitivity label specifies the lowest classification of data that may be stored in the table, and the sensitivity label must dominate the sensitivity label of the database.

Since tables contain multi-level data, tables are implemented using bundles (see page 131, "Bundles"). The bndlespace contains the sensitivity labels assigned to each tblspace in the bundle. A row's sensitivity label is retrieved from the bndlespace entry. Each row in the table must dominate the sensitivity label of the table.

The sensitivity label of a view definition is stored in the corresponding **systables** table entry of the system catalog. The view definition must dominate all base tables from which the view definition was derived.

The sensitivity label of a synonym must dominate the sensitivity label of the base table or view on which it depends. The **systables** table entry stores the synonym's sensitivity label.

The sensitivity label of a constraint must equal the sensitivity label of the table to which it is related. Like constraints, indexes can be considered as part of a table; therefore, the sensitivity label of an index must equal the sensitivity label of the table to which it is applied. For constraints, sensitivity labels are stored in the corresponding **sysconstraints** table entries. Sensitivity labels for indexes are stored the corresponding **sysindexes** table entries.

Label tranquility is preserved by only allowing the DBSSO to change the sensitivity label of an object. The DBSSO can only change a sensitivity label via the SAFE. The DBSSO cannot change an object's sensitivity label if another user is currently accessing the object. The SAFE requires an exclusive lock in order to change an object's sensitivity label; therefore, the DBSSO must wait until no users are accessing the object before the DBSSO changes the sensitivity label.

5.2.3 Object Reuse

Object reuse concerns the allocation of resources which have been used to store information and then released back to the system for future use. Subjects must not be able to scavenge data from resources previously allocated to other subjects. INFORMIX-OnLine/Secure's policy is to restrict access to an object until the resource has been written into. The RDBMS Kernel does not clear the resource in any way. Basically an object cannot be read by a user until it has been written.

Object reuse controls applied to disk and file pages, global shared memory buffers, databases, tables, rows, constraints, indices, views, synonyms, and BLOBs is discussed below.

Chunk and dbspace reserved pages, bitmap pages, chunk free extent pages, tblspace-tblspace pages, and BLOB free map pages are used only by INFORMIX-OnLine/Secure and are initialized by the RDBMS Kernel. Tblspace pages (row, index, and in-tblspace BLOB) are added to a tblspace when an extent is added to the tblspace. The extent pages are marked as free pages and can be used for any type of tblspace page. The pages cannot be read until they are used. When a page is needed, the page is marked used, the page header and footer are initialized, and the page contents are managed depending on what kind of page it is. BLOBspace-pages are treated in a similar manner. The pages in a BLOBspace are marked as free until needed. A BLOBspace-page is marked used and has its header and footer initialized when first used. See below for descriptions of object reuse prevention for row, index and BLOB objects.

Buffers no longer in use are placed on Least Recently Used (LRU) lists. Whenever a buffer is needed by the

Database Object	**Database Access**	**MAC Permissions**
Database	Create Database	sl(database) assigned sl(subject)
	Drop Database	sl(subject) = sl(database)
	Open Database	sl(subject) dom sl(database)
	Close Database	none
Table	Create Table	sl(table) assigned sl(subject)
	Drop Table	sl(subject) = sl(table)
	Open Table	sl(subject) dom sl(table)
	Close Table	none
	Alter Table	sl(subject) = sl(table)
	Rename Table	sl(subject) = sl(table)
Row	Insert Row	sl(row) assigned sl(subject)
	Delete Row	sl(subject) = sl(row)
	Read Row	sl(subject) dom sl(row)
	Modify/Update Row	sl(subject) = sl(row)
View	Create View	sl(subject) dom sl(base table)
		sl(view) assigned sl(subject)
	Delete View	sl(subject) = sl(view)
Synonym	Add Synonym	sl(subject) dom sl(table)
		sl(synonym) assigned sl(subject)
	Drop Synonym	sl(subject) = sl(synonym)
Constraint	Add Constraint	sl(subject) = sl(table)
		sl(constraint) assigned sl(subject)
	Drop Constraint	sl(subject) = sl(constraint)
Index	Add Index	sl(subject) = sl(table)
		sl(index) assigned sl(subject)
	Drop Index	sl(subject) = sl(table)

Table 5.3. MAC Permissions Required for Database Capabilities

RDBMS Kernel, the buffer at the end of one of the LRU lists is used. A request for reading or writing a page from disk causes the buffer contents to be overwritten with the requested page. A request for an empty page causes an unused buffer to be initialized with a page header and footer.

Creation of a database causes a new row to be entered in the database tblspace and causes the system catalog tables to be created. Deletion of a database causes all the tables and the system catalog in the database to be deleted. The row describing the database in the database tblspace is then deleted. Deletion and creation of tables and rows is described below.

Creation of a table involves building a bndlspace along with all its tblspaces. Memory structures in global shared memory are initialized, the table's tblspace-tblspace page is allocated, and the table's first extent is allocated. Deletion of a table causes the entry for the table in **systables** to be removed along with entries from other system catalog tables that refer to the table's columns, indexes, constraints, and privileges. All views that depend on the table are dropped. The bndlspace is then dropped by releasing any BLOBs associated with the table that reside in a BLOBspace and releasing all of the table's allocated extents.

When a row is deleted the actual data is not cleared. The slot table entry (see page 34, "Page Management") in the row's page is changed to indicate that the row has been deleted and the space cannot be accessed until a new row has overwritten that space. When space is needed to insert a new row in a table, a page (or pages) with enough room is located, the slot on the page is initialized, and the row's data is copied from kernel buffers onto the page. A new row's slot contains the length of a row so that when a new row is inserted into a page and does not overwrite all the data from a previous row, the old data cannot be accessed.

When a view is created a row in the **sysviews** and **sysdepend** system catalog tables are inserted and when a view is dropped the rows describing it are deleted from those tables. Reuse is based on the reuse applied to rows.

When a synonym is created a row in the **syssyntable** system catalog table is inserted and when a synonym is dropped the rows describing it are deleted from that table. Reuse is based on the reuse applied to rows.

When a constraint is created a row is inserted into the **sysconstraints** system catalog table and when a constraint is dropped the row describing it is deleted from that table. Reuse is based on the reuse applied to rows.

When an index for a table is created or when an existing index needs to expand, free pages are allocated from the bndlspace's tblspaces and initialized as index pages. When an index is dropped or when the index tree shrinks, the free list in the tblspace is changed to indicate the pages that are available.

Whenever a BLOB is deleted the row that contains the tuple-BLOB structure is updated to contain either NULL if the BLOB was simply deleted, or a new tuple-BLOB structure if the BLOB was replaced. Creation of a BLOB causes the tuple-BLOB structure to be inserted into the BLOB's row. Reuse of BLOB pages is handled differently depending on where the BLOB is stored. When a tblspace BLOB is deleted all the slots used by the BLOB are marked free in the same manner as for rows. If any of the slots occupy a whole page the status of the page is changed to free. When a tblspace BLOB is created, space is allocated in the same manner as for rows. The portions of the BLOB, other than the last, use all the space on a page as one slot. When a BLOBspace BLOB is deleted all the BLOBspace-pages are marked as free. When a BLOBspace BLOB is created a free BLOBspace-page is found, the BLOBspace-page is initialized, and the BLOB data is then written.

5.2.4 Auditing

INFORMIX-OnLine/Secure has the ability to audit all security relevant events that take place in INFORMIX-OnLine/Secure. The DBSSO chooses the events to be audited and maintains the audit masks. The audit masks indicate which events should cause an audit record to be created and inserted in the OS's audit log. The Audit Analysis Officer (AAO) can then extract all the INFORMIX-OnLine/Secure audit logs from the OS's audit log and perform audit analysis. See page 90, "Audit Analysis Officer" for a description of the tools an AAO uses to perform audit analysis. This section describes audit masks, lists the auditable events and the information recorded for those events, and describes audit record creation.

5.2.4.1 Audit Masks

An audit mask specifies a set of user events to be audited. The domain of auditable events is fixed; however, the DBSSO is responsible for choosing which events to audit. Audit masks are implemented as a sequence of bits, one for each auditable event. The masks are stored in the audit tblspace where each entry in the tblspace contains a user name and the user's associated audit mask (see page 34, "Tblspace Management". The audit tblspace is a RDBMS Kernel table that is kept in the root dbspace and is not part of any database.

The following describes the type of audit masks the DBSSO can use:

- Compulsory Mask - Events which are always audited for all users
- Individual User Mask - Events for a particular user that need to be audited
- Default Mask - Events which are audited for those users without an individual audit mask
- Template Mask - A pre-defined set of auditable events that may be used to quickly change another audit mask
- DBSA Mask - Events which are audited for all DBSAs

Each individual user always has two masks applied to their actions when using INFORMIX-OnLine/Secure. One is the compulsory mask, the other is either the default mask or an individual user mask.

The DBSSO maintains audit masks by using SAFE. SAFE allows the DBSSO to add, delete, and modify audit masks. For mask creation and modification the SAFE menus provide a list of auditable events to the DBSSO and the DBSSO chooses which events to audit. Creation of an audit mask causes a new entry to be added to the audit tblspace. Deletion of an audit mask causes the mask and its name to be removed from the audit tblspace. The compulsory, default, and DBSA audit masks cannot be deleted. When a user's audit mask is changed the changes take effect immediately. The SAFE can generate an audit mask report (listing audit mask contents and names) to help the DBSSO maintain the audit masks. The report is stored in an OS file specified by the DBSSO. The file obtains label (in B1/EA and B1/EP configurations), group, and DAC permissions so only that DBSSO can access the file.

5.2.4.2 Audited Events

INFORMIX-OnLine/Secure can audit security relevant events including administrator actions and object creation/deletion/access. The DBSSO has significant leeway when setting the audited events of INFORMIX-OnLine/Secure. Events always audited are DBSSO actions, INFORMIX-OnLine/Secure utility actions, and the start of a new RSAM session. The DBSSO does not need an audit mask because all DBSSO actions are always audited.

The format for INFORMIX-OnLine/Secure audit records as stored in the OS audit trail has two parts. The first part is an OS audit header. It contains information supplied by the OS. The second part of the audit record is generated by INFORMIX-OnLine/Secure. It contains a fixed field to identify the audit record as one generated by INFORMIX-OnLine/Secure, the session label of the process that performed the audited action, the real user ID of the process that performed the action, an indication of the success or failure of the action, the role of the user that performed the action (i.e., standard user, DBSA, DBSSO), the event code indicating the action performed, and additional fields depending on the action audited. The additional fields could include information like: an object's name, an object's label, an object's owner, or a user's privilege. For administrator actions the additional fields could include the command line of the transient process executed, the audit mask read or updated, or an object's old and new label. In both the AT&T and Harris systems, audit records are prefaced with OS headers. In the AT&T and Harris systems, the Process ID (PID) is included in the header and the PID can be traced to follow the actions of a specific user. In the HP system, the header includes the users login ID, which is never changes throughout the lifetime of a process. That login ID is used for individual accountability. The kinds of events audited and the types of additional field information include the following:

- user actions:
  - session startup: none
  - object creation/deletion/access: object name, object label, object ID, label at which object was opened
  - database privilege granting/revoking: database name, grantor, grantee, privilege, revokee
  - table privilege granting/revoking: table name, grantor, grantee, privilege, revokee
  - transaction management: none
- administrator actions:
  - chunk management: dbspace number, chunk number, mirror status
  - dbspace management: dbspace name, mirror status
  - BLOBspace management: BLOBspace name, mirror status
  - transaction log management: none
  - read/delete audit masks: audit mask name
  - create/update audit masks: audit mask name, audit mask
  - grant DAC privilege: object name, object label, privilege, grantor, grantee
  - revoke DAC privilege: object name, object label, privilege, revoker, revokee
  - access a database: database name, label
  - database label modification: database name, old label, new label
  - table access: database name, table id, table label
  - table label modification: database name, table name, old label, new label, owner
  - use of transient processes: command line used when process was executed

5.2.4.3 Audit Record Creation

When a user session is initialized the RDBMS Kernel creates a current audit mask in the entry in the User Table of global shared memory. The current audit mask is the union of all applicable audit masks at session startup. For standard users the individual audit mask is OR'd with the compulsory audit mask. If the user does not have an individual audit mask the default mask is used instead. All DBSAs use the DBSA audit mask.

The RDBMS Kernel is structured so that audit record creation takes place based on the return value of routines. Before a RDBMS Kernel routine returns its results to the user, the RDBMS Kernel checks the current audit mask to see if an audit record should be created. Then the audit record is created using the values stored in the User Table, object identifiers, the return value of the event, and other event specific information.

The audit record created is then passed to the OS to be added to the OS audit log. How the audit record is placed in the OS audit log depends on the interface provided by the OS (see page 11, "Informix in the Operating System Environment"). INFORMIX-OnLine/Secure relies on the OS audit log mechanisms for protection and maintenance of its audit records.

5.3 Administrative Roles

This section describes the INFORMIX-OnLine/Secure privileged administrator roles: the Database System Security Officer (DBSSO) is described on page 88, "Database System Security Officer and Secure Administrator Front End", the Database System Administrator (DBSA) is described on page 89, "Database System Administrator", and the Audit Analysis Officer (AAO) is described on page 90, "Audit Analysis Officer". Administration of INFORMIX-OnLine/Secure also requires functions to be performed by an administrator of the OS. The role of the Operating System Administrator (OSA) is discussed in the next section.

5.3.1 Operating System Administrator

The Operating System Administrator (OSA) is more a collection of responsibilities and tasks that INFORMIX-OnLine/Secure requires from the operating system than a "role". In other words, the tasks that need to be performed in support of INFORMIX-OnLine/Secure by someone administrating the operating system are grouped into this role. The OSA has numerous responsibilities related to the installation and set up of INFORMIX-OnLine/Secure. The OSA has continued responsibility of granting and revoking access to and from INFORMIX-OnLine/Secure, adding new DBSSO and DBSA accounts as necessary, and coordinating with the DBSSO about various security-related aspects of the system.

The OSA is largely responsible for the INFORMIX-OnLine/Secure system installation and that is discussed in the following paragraphs. Before unloading the INFORMIX-OnLine/Secure software from the distribution media, the OSA must define new mandatory and discretionary access control privileges. The OSA must create new UNIX groups as shown in Table 5.4. In addition, during installation the OSA must also create the MAC categories of **IX_DBSSO, IX_DBSA, and IX_DATA** for the EA and EP configurations. The OSA is responsible for the sensitivity labels and may create a **Syslo** in the OS which represents the lowest level in the operating system at which no users are allowed to write. **Syslo** is used as a MAC means of preventing any user from tampering with the executable code; otherwise, executables are protected by DAC only. The Informix TFM suggests (but does not require) that **Syslo** be established on the target OS. If this is done, then the executables are protected by DAC and MAC. However, if this in not done, the only protection is then DAC.

The OSA must create a directory pointed to by the environment variable **$INFORMIXDIR**, which contains multiple instantiations of the system software for INFORMIX-OnLine/Secure. In B1 systems, the

New Group	Members
ix_data	Only root
ix_dbsso	Only accounts for DBSSO use and root.
ix_dbsa	Only accounts for DBSA use and root.
ix_users	All users who want to use INFORMIX-OnLine/Secure, and root

Table 5.4. New UNIX Groups

$INFORMIXDIR directory should be created with the appropriate security permissions, labels or privileges. In addition, the OSA must create a directory to hold files created by the DBSA utility programs for monitoring and tuning purposes. The files have the label of **DataHi+IX_DATA**, and are in the group **ix_data**.

The OSA must create special user accounts before the installation of INFORMIX-OnLine/Secure can take place. The designated user accounts are a DBSA account, a DBSSO account, the "informix" account, and an AAO account.

5.3.2 Database System Security Officer and Secure Administrator Front End

The Database System Security Officer (DBSSO) role is responsible for performing all the routine tasks related to maintaining the security of an INFORMIX-OnLine/Secure system. These tasks are:

- Maintaining the audit configuration.
- Reviewing the audit trail (or overseeing the review).
- Modifying labels of OnLine/Secure objects, when necessary, in B1 systems.
- Modifying OnLine/Secure discretionary access control, when necessary.
- Participate in the Installation and Set-up of INFORMIX-OnLine/Secure.
- Diagnosing and responding to security problems.
- Educating users.

The DBSSO's tasks should be performed by someone who has the appropriate clearance to view all the data in the database. It is necessary for the individual performing this role to have a high enough clearance to view all the data in the system because they are responsible for the security of items at all levels and make decisions about reclassification and audit configuration. A user is a DBSSO because their account is a member of the group **ix_dbsso** and, in the B1 configurations, has the category **IX_DBSSO**. The DBSSO role is supported by software (the Secure Administrator Front End, SAFE) which only provides those functions that the DBSSO must perform. In order to use the SAFE, the person filling the DBSSO role must log into the designated user account. Actions performed by the DBSSO are always audited. In addition, when acting as DBSSO, the only software that can be run is the SAFE. The SAFE utility is only invoked by the DBSSO and is protected by OS DAC (**ix_dbsso**) and MAC (**DataHi+IX_DBSSO**). The DBSSO may not perform regular user activities while logged in as the DBSSO; see page 39, "Relational Storage Access Method (RSAM) Processes" for a discussion of the SAFE interface.

The DBSSO has significant leeway when setting the audit level of INFORMIX-OnLine/Secure. The minimal audit level is audit of DBSSO actions and the starting of each new INFORMIX-OnLine/Secure session.

INFORMIX-OnLine/Secure also suggests that the following events be audited in conjunctions with the minimal activities: Grant DB Access, Revoke DB Access, Grant Table Access, and Revoke Table Access. Finally, during the installation of INFORMIX-OnLine/Secure the DBSSO is responsible for the initial audit configuration.

Occasionally, in B1 systems, it may be necessary to modify the sensitivity label associated with an INFORMIX-OnLine/Secure storage object. The DBSSO is the only user who has the authority and the capability to modify these labels. Therefore the DBSSO is exempt from the Informix MAC restrictions and the DBSSO can access all objects between **Datahi** and **Datalo**, and change object labels to any label between **Datahi** and **Datalo**. For example, a user may have imported some data into a database at a higher level than was necessary. The DBSSO can then downgrade the database object to accurately reflect its proper classification.

The specific INFORMIX-OnLine/Secure storage object sensitivity labels which the DBSSO can modify are: Databases, Tables, and Rows. It is not possible to directly modify the sensitivity labels of constraints, and indexes because those labels are modified when their associated table, or row is modified. Synonyms and view labels cannot be modified. When the labels of a table are modified and these table labels strictly dominate the labels of the views, all the corresponding views and synonyms are dropped.

Standard INFORMIX-OnLine/Secure users can grant and revoke discretionary privileges for INFORMIX-OnLine/Secure objects according to the discretionary privileges that they themselves possess, and whether or not they are the creator of the object. The DBSSO is exempt from the discretionary access control restrictions imposed upon standard INFORMIX-OnLine/Secure users. Therefore, the DBSSO has the authority to grant and revoke the discretionary privileges associated with any of these objects within INFORMIX-OnLine/Secure to or from any INFORMIX-OnLine/Secure users. In addition, the DBSSO cannot modify the accesses on a database if a user has the database locked.

5.3.3 Database System Administrator

The Database System Administrator (DBSA) role is a privileged role responsible for configuring, tuning, and monitoring INFORMIX-OnLine/Secure. These activities are necessary to keep the system running efficiently and are not involved directly with the security of the system. The DBSA becomes involved with the security of INFORMIX-OnLine/Secure during installation and when importing or exporting labeled data to or from INFORMIX-OnLine/Secure.

There are no special account names reserved for performing administrative tasks. Instead, in the EA and EP configurations, mandatory and discretionary controls are used to restrict access to DBSA accounts. In the C2 configuration, only discretionary controls are used. In the B1 configurations, the **IX_DBSA** category is created during system installation. This category must be in the clearance assigned to any user authorized to perform the DBSA role. In all configurations, the **ix_dbsa** group is created during system installation. Accounts authorized to perform the DBSA role must be able to log in as a member of the **ix_dbsa** group. While logged in to an account that is in the **ix_dbsa** group, the DBSA is not allowed to access INFORMIX-OnLine/Secure as a regular user; the system can only be accessed through the Administrative Front End (AFE). The trusted command-line utilities are: *tbcheck*, *tbinit*, *tbload*, *tblog*, *tbmirror*, *tbmode*, *tbparams*, *tbspaces*, *tbstat*, *tbtape*, and *tbunload*. See page 60, "Support and Transient Processes" for a discussion of these utilities and the support processes that they spawn.

Only the DBSA is capable of exporting labeled INFORMIX-OnLine/Secure objects. When the DBSA performs one of the following tasks, INFORMIX-OnLine/Secure data is exported to a secondary storage media along with its associated non-advisory sensitivity label:

- Archiving using *tbtape*,
- Backing up the logical logs using *tbtape*, or
- Writing a database or table to secondary storage media using *tbunload*.

The DBSA can import labeled data into an INFORMIX-OnLine/Secure object from a magnetic media by either restoring INFORMIX-OnLine/Secure from the archive tape and the logical logs (using *tbtape*), or reading a database or table exported with *tbunload* (using *tbload*). When the data is imported in one of these two ways, the sensitivity labels associated with the data are read in along with the data, independent of the DBSA session sensitivity label. If the sensitivity labels of the data to be loaded are the same as those on the exporting system, the DBSA can perform the load with no problems. If, however, the sensitivity labels on the export system are either undefined on the import system or differently defined, the DBSA requires the DBSSO to manage the label discrepancies. The DBSSO must create the file **label.map** which contains the mappings between the two sets of labels (see page 12, "Required OS Services" for additional discussion of the **label.map** file).

5.3.4 Audit Analysis Officer

INFORMIX-OnLine/Secure provides the ability to read audit trail data into one of its own databases, so that it can be analyzed using SQL. Since the DBSSO and DBSA roles cannot perform standard user activities, like using SQL, a new role was created to perform audit analysis, i.e., the Audit Analysis Officer (AAO). The AAO has access to view all the data in INFORMIX-OnLine/Secure. In the EA and EP configurations, a user authorized to perform AAO activities must be cleared to **Datahi**. In all configurations, the account authorized to perform AAO activities must be able to log in as a member of the **ix_users** group.

As described on page 85, "Auditing", INFORMIX-OnLine/Secure obtains the information about events that are being audited and forwards that information to the OS for recording. The OS audit trail consists of both OS and INFORMIX-OnLine/Secure audit records. Before the AAO can analyze the INFORMIX-OnLine/Secure audit trail, the DBSSO extracts the INFORMIX-OnLine/Secure audit records from the OS audit trail using *datextract*. This utility allows database-specific audit records to be extracted from the Os audit trail. The *datextract* utility also provides an option that allows audit records to be extracted by sensitivity label. The output file from this operation is protected by setting the owner of the file equal to the AAO account, the group equal to **ix_users**, and, on the B1 configurations, setting the sensitivity label to **Datahi**. The file permissions are set to read permission only at creation time for the owner. In this way, the extracted INFORMIX-OnLine/Secure audit records are accessible only to the AAO.

Once the INFORMIX-OnLine/Secure audit records are available to the AAO, the audit data can be viewed with the *datconfig* utility which displays the records on the terminal screen formatted for easy viewing.

Chapter 6

Assurance

6.1 Rating Maintenance Phase

After the Formal Evaluation Phase, Informix will enter the Ratings Maintenance Phase (RAMP). They have identified a Vendor Security Analyst (VSA), a Responsible Corporate Officer (RCO) who is the Vice-President of Quality and Standards, and Vendor Point of Contact for business issues.

The Informix approach to RAMP mirrors that of a stop light. The three types of changes are: RED - security critical; YELLOW - security relevant; and GREEN - non security relevant. The type of change determines what evidence needs to be presented for a successful RAMP action. For a RED change, a VSA would be required to present RAMP evidence to a Technical Review Board (TRB). A YELLOW change would also require the TPOC present evidence to the TRB, but probably fewer details than with a RED change. A GREEN change would require the TPOC to review the evidence and possibly a mail-in TRB.

Each RAMP may be made up of several RAMP components each having a different color; components of the RAMP are software modules and are handled as described above. The color of a RAMP component is determined by two factors: the color of the module changed and the scope of the change. The following matrix outlines the possible color combinations for each component and their resulting RAMP component color classifications. See figrefcolors for a graphical representation of the possible color combinations.

In the RM-Plan [17] there is a listing of all TCB modules, and their associated colors. Informix will justify module color classifications when a specific module is presented for a RAMP action.

6.1.1 Configuration Item Identification

The INFORMIX-OnLine/Secure software is decomposed into high-level configuration items. This software decomposition is used to precisely specify the configuration of the system at discrete points. The CCC Softool product, from the Softol Corporation, is used to keep track of files that make up the system. The configuration items are listed below.

- TCB source files
- the software used to develop and validate the TCB as described in the *Software Development Plan.*
- *Security Policy Model*
- *INFORMIX-OnLine/Secure Security Features User's Guide Database Server Version 4.1*
- *INFORMIX-OnLine/Secure Trusted Facility Manual Database Server Version 4.1*
- *Security Test Plan for INFORMIX-OnLine/Secure* and *Security Test Procedures for INFORMIX-OnLine/Secure*
- the set of design documentation
- *RM-Plan for INFORMIX-OnLine/Secure*

		Scope of Change		
		Red	Yellow	Green
Module Sensitivity	Red	Red	Red	Red
	Yellow	Red	Yellow	Yellow
	Green	Red	Yellow	Green

Figure 6.1. Determining the color classification of the RAMP components

6.1.2 Configuration Management Process

A request for enhancement starts the product development cycle depicted in Figure 6.2. Such a request can necessitate a document or feature change, or a bug fix. Following is a description of each phase in the product development cycle. INFORMIX-OnLine/Secure goes through this development cycle on one OS, and then is ported to others.

The request for product/enhancement phase includes a gathering data phase. Once the available data is analyzed, specific feature or bug fix requests are identified. Data may be gathered from customers or from a feature request database. This database is maintained by Informix and consists of suggestions from every branch of Informix. Once consensus is reached that there is a need for the new release, the product feasibility stage begins. During this short phase, a general definition of the content and scheduling of the product are developed.

The product proposal phase begins with the creation of the product team. It is in this phase that the justifications for the product are finalized. Once the plans are formalized and approved, the product development team begins the process of creating the functional specification. After the functional specification has been developed, the product moves into the planning and design phase.

It is in the planning and design phase that the VSA first performs security analysis. Security analysis involves ensuring that the change does not affect the database security mechanisms, as well as the OS security policy. In determining whether the OS policy has not been altered, the VSA must analyze the OS interface and determine how the change affects that interface. Additionally, a design specification called a Checklist of Deliverables, and a Beta Testing Plan are generated.

The development effort proceeds from the Checklist of Deliverables and proceeds through several phases through the final configuration. The phases the product passes through are: Pre-Alpha, Pre-Beta, Beta, and Post Beta. After the product has completed development, it goes before a Configuration Review Board (CRB) for final approval. The result of the CRB is a certification report that is sent to NSA, and internal reports used in tracking changes.

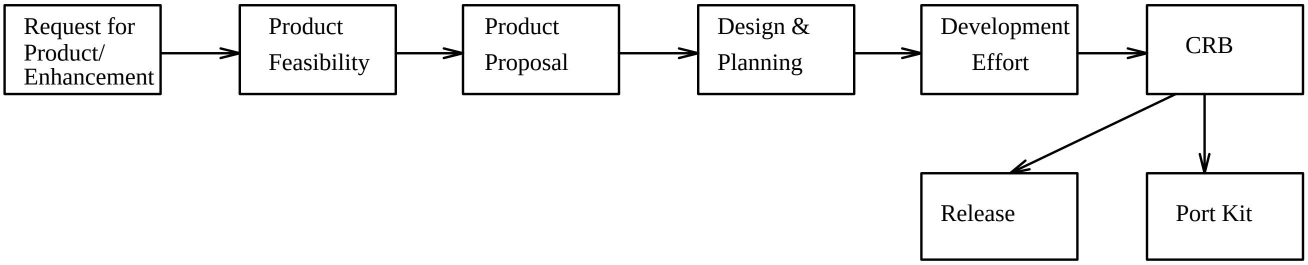


Figure 6.2. The product development cycle

Additionally, the product may be released along with a Port Kit. The Port Kit consists of a source tape and a test suite. The following process takes place during a porting effort. The first step is an analysis of the secure OS TCB including a review of the OS documentation to ensure the OS security policy is compatible with the the *Security Policy Model* for INFORMIX-OnLine/Secure. In performing this evaluation, the VSA must provide written evidence that all system calls to the OS interface were examined and a complete understanding of the privileges of each system call was reached. An understanding in this sense means Informix will analyze the constraints that a trusted process must adhere to while operating with privilege in the OS environment. The next step is to determine whether the desired change is a port to a new system or an upgrade to a previously evaluated/ported system. If the port is an upgrade, then a brief report is sent to NSA justifying the determination. If it is not an upgrade, the Configuration Control Board is convened to evaluate the security implications of the proposed changes. It is important to note that Informix does not configuration manage changes to the operating system. Instead, if the operating system performs a RAMP action , Informix performs a port onto the RAMP version of the operating system. After the security analysis has been performed, the Port Kit is given to the porting division to make the necessary modifications to INFORMIX-OnLine/Secure. The security test suite is executed at the end of each RAMP cycle.

6.2 Security Testing

The goal of the INFORMIX-OnLine/Secure Security Test Suite is to provide assurance that the security aspects of INFORMIX-OnLine/Secure work as claimed. These tests encompass security-relevant features only, and do not test non-security-relevant functional features. The Security Test Suite comprises five formal tests which are designed to test the mechanisms used to enforce the security policy. The five formal tests are:

- User Identification Tests,

- Sensitivity Label Tests (EA and EP only),
- Mandatory Access Control Tests (EA and EP only),
- Discretionary Access Control Tests, and
- Audit Tests.

In addition, object reuse is tested as a side effect of other tests. Most object reuse tests attempt to access rows of a table by rowid after the row has been deleted, and so on. Object reuse tests are spread throughout the other tests as are system architecture tests. Most system architecture tests involve attempting to access the database, tables and rows via OS primitives rather than RDBMS primitives.

Each of the five formal tests in the INFORMIX-OnLine/Secure Security Test Suite are associated with two test classes: a valid input test class and an erroneous input test class. Since it is virtually impossible to test all possible input combinations, both test classes test a representative sample of inputs including maximum, minimum, and mid-range values. The valid input test class is designed to verify the ability of INFORMIX-OnLine/Secure to recognize and properly respond to acceptable input values. The erroneous input test class is designed to test the robustness of INFORMIX-OnLine/Secure's recovery mechanisms and to verify that INFORMIX-OnLine/Secure operates in the prescribed manner for all erroneous input values. INFORMIX-OnLine/Secure tests are conducted using one or more of the following methods:

- Inspection (I) - Examination of code or review of design documentation to confirm compliance with a specific requirement(s).
- Analysis (A) - Review or interpretation of analytical or empirical data under defined conditions or reasoning to show theoretical compliance with a specific requirement(s).
- Demonstration (D) - Verification of an operational or functional capability by performance. Compliance with requirements is determined by observation of the software operation or inspection of the output data, or both.
- Test (T) - Performance of functional operation under specified conditions and involving the use of special test equipment or software to generate, acquire, and record test data. Verification of compliance with requirements is determined by observation of operations or inspection of output data.

Each formal test comprises one or more test cases. For each test case, the test description includes a requirements traceability which identifies all INFORMIX-OnLine/Secure functions tested in the test case and maps each function to the Verification Cross Reference Matrix (VCRM) found in the *Security Test Plan for INFORMIX-OnLine/Secure*. The VCRM in turn maps each function being tested to a particular security requirement. Documentation is provided which identifies where additional documentation can be found about the design of the function which implements each security requirement. In this way, Informix provides complete coverage of the security-relevant functions of INFORMIX-OnLine/Secure.

6.3 Model

Informix has produced an informal model, *Informal Security Policy Model for INFORMIX-OnLine/Secure* [19], of the security policy enforced by INFORMIX-OnLine/Secure. The model is formulated in two parts, one for the OS and one for the database management system itself. There is also a description of the combination of the two parts which highlights security properties of the combination.

The model is based on a state machine model like the model developed by Bell and LaPadula [1]. The OS model describes the security policy the OS must meet. The DBMS model describes the security policy applied to INFORMIX-OnLine/Secure subjects and objects. The composite model is constructed so that its security properties follow directly from the security properties of the other two models.

Each model describes a set of states, a set of state transitions, a set of axioms (representing assumptions of the model), and a set of properties (representing the policy to be enforced). The descriptions are based on set theory and predicate logic. The descriptions address users, subjects, objects, MAC, DAC, states in system, the initial state, and state transitions.

This page intentionally left blank

Chapter 7

Evaluation as a C2 System

This chapter describes how the TCSEC requirements and the TDI interpretations are satisfied by INFORMIX-OnLine/Secure executing on top of operating systems in the evaluated configuration. The rating earned by Informix will be associated with the composite TCB comprising INFORMIX-OnLine/Secure and either System V/MLS&T System V/MLS, Harris CX/SX, Harris CX/SX with LAN/SX, or HP-UX BLS. Consequently, the requirements are considered from the perspective of the composite TCB. As described throughout this report, INFORMIX-OnLine/Secure depends heavily on the operating system security mechanisms to enforce its portion of the composite security policy; therefore the explanation of how some requirements are met lies in how the evaluated operating system meets the requirements. This chapter does not address the details of how the evaluated operating system meets the TCSEC requirements, but addresses what INFORMIX-OnLine/Secure explicitly relies on to enforce its portion of the policy. For details concerning each operating system design and implementation, see each of the Final Evaluation Reports [[5], [6], [7]].

7.1 Discretionary Access Control

Requirement

TCSEC:

The TCB shall define and control access between named users and named objects (e.g., files and programs) in the ADP system. The enforcement mechanism (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of those objects by named individuals, or defined groups of individuals, or by both, and shall provide controls to limit propagation of access rights. The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorized access. These access controls shall be capable of including or excluding access to the granularity of a single user. Access permission to an object by users not already possessing access permission shall only be assigned by authorized users.

TDI

The discretionary access control requirements apply as stated in the TCSEC to every TCB subset whose policy includes discretionary access control of its subjects to its objects. Any TCB subset whose policy does not include such discretionary access control is exempt from this requirement.[1]

[1] Note that any evaluation by parts requires that at least one TCB subset in the TCB enforce a discretionary access control policy, and thus satisfy this requirement.

Applicable Features

The OS provides DAC on named objects under its control such as files and devices. The INFORMIX-OnLine/Secure DAC mechanism is distinct from that of the OS; it replaces the OS DAC policy within the RDBMS by applying access attributes specific to RDBMS named objects, rather than just the read, write, and execute control placed on OS named objects. The RDBMS named objects correspond to databases, tables, views, synonyms, constraints, and indexes. DAC is accomplished via privileges which users grant and revoke using SQL statements or the RSAM interface. Privileges are granted to single users by name or to all users under the name of PUBLIC.

When a new named object is created, the *initial access* to it is established by default. If the named object is a database, the user who creates the database is given the **dba** privilege. If the named object is a table, the creating user becomes the owner and is given all table level privileges including the permission to grant and revoke these privileges to and from other users. If the named object is a view, synonym, constraint, or index, the creating user becomes the owner of the object but gets no specific privileges since none are defined for these types of named objects. No user other than the creator initially possesses any privileges to a named object with the exception of the Database Administrator (called the dba) who has implicit privileges on the tables in the database.

Changes in the discretionary access of a user to a named object can occur when a subject explicitly grants or revokes privileges to or from the user to the named object. A subject cannot grant or revoke a database level privilege (**dba**, **resource**, **connect**) to or from another user unless the subject possesses the **dba** privilege. A user, other than the table owner, possesses a table level privilege only if someone else previously granted that user the privilege. Each table level privilege can have a *grant* option associated with it which allows the user to give the privilege to another user.

A user can only revoke a table level privilege from another user if the revoking user originally granted the table level privilege. However, table level privileges can be indirectly revoked from a user through a chain of cascading effects. In addition, INFORMIX-OnLine/Secure prevents cycles of grants. In this way, INFORMIX-OnLine/Secure limits the propogation of access rights.

Conclusion

INFORMIX-OnLine/Secure satisfies the C2 Discretionary Access Control requirement.

7.2 Object Reuse

Requirement

TCSEC:

All authorizations to the information contained within a storage object shall be revoked prior to initial assignment, allocation or reallocation to a subject from the TCB's pool of unused storage objects. No information, including encrypted representations of information, produced by a prior subject's actions is to be available to any subject that obtains access to an object that has been released back to the system.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset in the TCB.

Applicable Features

INFORMIX-OnLine/Secure doesn't provide any interfaces to an object's resources until the object has been used. An object cannot be read by a user until it has been written into. Resources allocated for an object's use cannot be accessed. The OS enforces an object reuse policy on its objects. See page 82, "Object Reuse" for details regarding object reuse for each object.

Conclusion

INFORMIX-OnLine/Secure satisfies the C2 Object Reuse requirement.

7.3 Identification and Authentication

Requirement

TCSEC:

The TCB shall require users to identify themselves to it before beginning to perform any other actions that the TCB is expected to mediate. Furthermore, the TCB shall use a protected mechanism (e.g., passwords) to authenticate the user's identity. The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user. The TCB shall be able to enforce individual accountability by providing the capability to uniquely identify each individual ADP system user. The TCB shall also provide the capability of associating this identity with all auditable actions taken by that individual.

TDI:

This requirement applies as stated in the TCSEC to the entire TCB. The cooperative action of the TCB subsets making up the TCB must satisfy the requirement.

If the TCB is composed of TCB subsets, one TCB subset may either rely on a mechanism in another TCB subset to provide identification and authentication services or provide the services directly. Each TCB subset may maintain its own identification and authentication data or one central repository may be maintained. If each TCB subset has its own data, then the information for each individual user must be consistent among all subsets.

Applicable Features

The OS in the evaluated configuration require users to explicitly identify themselves to the TCB with a unique user identifier and to authenticate themselves with passwords before they may access TCB protected resources. Each OS maintains clearance and authorization information for each individual to ensure that users may not invoke or access any data which is protected by labels outside their clearance range. The OS protects this identification and authentication information, making it accessible by OS administrative personnel only. Each audit record contains a unique user ID which is assigned to a user's process at login and is recorded in the audit records generated by all processes invoked by that user.

INFORMIX-OnLine/Secure relies solely on the underlying operating system to provide identification and authentication services; therefore identification and authentication information for the database and the operating system is kept in a central repository protected by the operating system. Although INFORMIX-OnLine/Secure generates its own audit records, it relies on operating system services to provide user identification information to uniquely associate auditable events with individuals. To ensure that user's may not invoke or access information outside their access range, INFORMIX-OnLine/Secure relies on the operating system to properly restrict users from logging into levels or groups to which they do not have access. Specifically, untrusted users are not allowed to belong to the group **ix_data**.

Conclusion

INFORMIX-OnLine/Secure satisfies the C2 Identification and Authentication requirement.

7.4 Audit

Requirement

TCSEC:

The TCB shall be able to create, maintain, and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorized for audit data. The TCB shall be able to record the following types of events: use of identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the object. The ADP system administrator shall be able to selectively audit the actions of any one or more users based on individual identity.

TDI

This requirement applies as stated in the TCSEC to the entire TCB. The cooperative action of the TCB subset making up the TCB must satisfy the requirement.

A TCB subset may maintain its own security audit log, distinct from that maintained by more primitive TCB subsets, or it may use an audit interface provided by a different TCB subset allowing the audit records it generates to be processed by that TCB subset.

If the TCB subset uses different user identifications that a more primitive TCB subset, there shall be a means to associate audit records generated by different TCB subsets for the same individual with each other, either at the time they are generated or later.

Auditable events, in the case of a database management system, are the individual operations initiated by untrusted users (e.g., updates, retrievals, and inserts) not just the invocation of the database management system. The auditing mechanism shall have the capability of auditing all mediated accesses which are visible to users. That is, each discretionary access control decision and each mandatory access control policy decision shall be auditable. Individual operations performed by trusted software, if totally transparent to the user, need not be auditable. If the total audit requirement is met by the use of more than one audit log, a method of correlation must be available.

Applicable Features

The RDBMS Kernel creates audit records of events and sends the records to the OS audit log using the interface provided by the OS. The RDBMS Kernel relies on the protection of the OS audit log for protection of RDBMS audit records. The RDBMS Kernel also relies on the OS audit mechanism implementation to limit the number of audit records that can be lost during system failures. The audit records stored in the OS audit log can be extracted to a file that only contains RDBMS Kernel audit records. This file is protected so that only the DBSSO or AAO can access the audit data.

The RDBMS Kernel has the ability to audit object creations and deletions, object accesses, object updates, invocation of the RDBMS Kernel, the use of locks, and the granting and revoking of DAC privileges. DBSA actions can also be audited including the use of support processes and changes to the RDBMS Kernel configuration. DBSSO actions such as maintenance of audit masks and maintenance of DAC privileges are always audited. The OS audits identification and authentication and all other OS specific events.

The information recorded by the RDBMS Kernel for each event includes the real user ID of the process that performed the event, success or failure of the event, and the event code. The OS audit mechanism supplies an audit header that contains the date and time of the event. An object's name is included in events that create, delete, access, update, or in any way relate to an object.

The DBSSO can selectively audit on a per user basis using audit masks. A system wide default of auditable events can also be set up. See page 85, "Auditing" for details on the audit mechanism.

Conclusion

INFORMIX-OnLine/Secure satisfies the C2 Audit requirement.

7.5 System Architecture

Requirement

TCSEC:

The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g., by modification of its code or data structures). Resources controlled by the TCB may be a defined subset of the subjects and objects in the ADP system. The TCB shall isolate the resources to be protected so that they are subject to the access control and auditing requirements.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset, with the following additional interpretations.

The TCB must provide domains for execution that are allocated to and used by TCB subsets according to the subset-domain condition for evaluation by parts. A most primitive TCB subset must provide domains for execution. A less primitive TCB subset must make use of domains provided by a more primitive TCB subset. A less primitive TCB subset may provide further execution domains but is not required to do so.

If the TCB is composed of multiple TCB subsets, this requirement applies to each TCB subset.

Applicable Features

The OS TCB subset implements a domain for its execution that protects it from external interference or tampering as well as process isolation through distinct process address spaces. It protects resources it controls and subject those resources to access control and auditing requirements. Details of the each of the operating systems' domain and process isolation mechanisms can be found in the Final Evaluation Report for each operating system [[5], [6], [7]].

The INFORMIX-OnLine/Secure TCB subset relies on OS mechanisms and services to maintain its domain isolation and protect it from external interference or tampering. Specifically, INFORMIX-OnLine/Secure executes as a trusted process in the context of the operating system and uses the operating system's process isolation properties to maintain and protect its execution domain. INFORMIX-OnLine/Secure uses the OS's DAC mechanisms to isolate its data structures (i.e., global shared memory and disk data structures) and subjects the resources it controls to access control and auditing policies. Specifically, OS objects containing RDBMS information are protected by traditional UNIX protection bits with access granted only to members of the **ix_data** group. No untrusted users are members of this group. INFORMIX-OnLine/Secure relies upon the OS process isolation coupled with the proper management of its resources, as described in Chapter 3, to provide each user a distinct address space. All INFORMIX-OnLine/Secure executables are protected from tampering using traditional UNIX protection bits with execute only permission granted to RDBMS users.

Conclusion

INFORMIX-OnLine/Secure satisfies the C2 System Architecture requirement.

7.6 System Integrity

Requirement

TCSEC:

Hardware and/or software features shall be provided that can be used to periodically validate the correct operation of the on-site hardware and firmware elements of the TCB.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset that includes hardware or firmware. Any TCB subset that does not include hardware or firmware is exempt from this requirement.

Applicable Features

This requirement is not applicable to INFORMIX-OnLine/Secure and is handled completely in the OS.

Conclusion

INFORMIX-OnLine/Secure satisfies the C2 System Integrity requirement.

7.7 Security Testing

Requirement

TCSEC:

The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. Testing shall be done to assure that there are no obvious ways for an unauthorized user to bypass or otherwise defeat the security protection mechanisms of the TCB. Testing shall also include a search for obvious flaws that would allow violation of resource isolation, or that would permit unauthorized access to the audit or authentication data.

TDI:

This requirement applies as stated in the TCSEC to the entire TCB. If a TCB consists of TCB subsets meeting the conditions for evaluation by parts, the satisfaction of the requirements by each TCB subset satisfies the requirement for the entire TCB. Otherwise, security testing of the entire TCB must be performed (even if the results of testing the individual TCB subsets were available).

Applicable Features

In addition to a test readiness review, the team verified the test tools that were used to run the test suite to ensure that they were operating properly. This included verifying the SAFE viewing mechanism and audit record generation based on a single audit flag.

The evaluation team observed and participated in the manual execution of the Informix test suite on all three configurations: C2, B1/EA, and B1/EP. Although this task was tedious, the procedures were easy to follow and the results were as expected. Several inconsistencies were uncovered during the first week of testing but were re-tested and resolved during the second week of testing.

Of particular concern was the fact that all testing was conducted at the interface to the SQL Engine. Since the RSAM interface is also part of the TCB interface in the B1/EA configuration, it was necessary to ensure that all RSAM interfaces were covered by the tests. To do this, all tests were run with a slightly modified RSAM that dumped all RSAM interface calls (in the form of RSAM messages) to a file. The team was able to examine this file to ensure complete coverage of the RSAM interface. Some RSAM interfaces, however, cannot be invoked directly via SQL commands. To test these interfaces, Informix provided a special menu-driven tool that allowed the tester to chose the interface to be tested and subsequently performed both a success and failure test. After each test was run, the audit log was examined and all test results were as expected. The team did not perform testing of the OS portion of the TCB. Instead, the team performed analysis of the RDBMS-OS interface and conducted tests to support the assertion that INFORMIX-OnLine/Secure does not inappropriately modify or corrupt the OS portion of the TCB. The team chose to perform more in-depth analysis at the RDMBS-OS interface rather than test the entire TCB because the team felt this approach provided more assurance. To conduct testing of the interface, the team identified security relevant OS files to monitor throughout the testing exercise. These files were backed up to tape and periodically during testing, the current state of these files was compared to the backup copy. These files were checked on each of the three testing machines with no inconsistencies found. For each machine, the team also periodically reviewed the OS audit trail for suspicious behavior by INFORMIX-OnLine/Secure. Specifically, the team looked for unexplained object reads and writes from test users on whose behalf INFORMIX-OnLine/Secure was executing. No unexplained actions were uncovered. A number of team tests were also run including passing invalid messages to RSAM; attempting to access global shared memory from an unprivileged user account; object reuse tests such as attempting to access a row that has been deleted and attempting to read rows from an empty table; and attempting to perform normal user activities while logged in as the DBSSO or DBSA. Several other team tests were also performed and all results were as expected.

Conclusion

INFORMIX-OnLine/Secure satisfies the C2 Security Testing requirement.

7.8 Security Features User's Guide

Requirement

TCSEC:

A single summary, chapter, or manual in user documentation shall describe the protection mechanisms provided by the TCB, guidelines on their use, and how they interact with one another.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset in the TCB. This collection of guides must include descriptions of every TCB subset in the TCB and explicit cross-references to other related user's guides to other TCB subsets, as required. In addition, interactions between mechanisms within different TCB subsets must be clearly described.

Applicable Features

The *INFORMIX OnLine/Secure Security Features User's Guide Database Server Version 4.1* is intended for users and provides descriptions of the functions of INFORMIX-OnLine/Secure.

The preface and the introduction of this document describes the contents of the manual and the proper use of the manual. In addition, these sections point to other reference materials that will help the user understand general security concepts. Finally, the introduction describes the different user roles and system configurations.

Chapter 1 of the SFUG is entitled "Security Concepts" and provides an overview of the security mechanisms utilized by INFORMIX-OnLine/Secure, as well as some security terminology.

Chapter 2, "Using INFORMIX-OnLine/Secure in a Multilevel Environment," describes how B1 INFORMIX-OnLine/Secure server is different from the general comercial product. It includes topics like multilevel tables, locking in a multilevel environment, how SQL statements behave in a multilevel environment, syntax and use of label comparison functions, and assorted session configuration information.

Chapter 3, "Using INFORMIX-OnLine/Secure in a Single-Level Environment," explains how a C2 INFORMIX-OnLine/Secure server is different from the comercial product. It discusses disk management of single-level tables.

Chapter 4, "Using INFORMIX-OnLine/Secure in Any Environment," contains information that users of either a C2 or B1 INFORMIX-OnLine/Secure server should know. It covers how some of the privilege-related functions differ from INFORMIX-OnLine/Secure, effective checking on unique constraints, parallel sorting for multiprocessor machines, and how utilities function.

Chapter 5, "Import/Export," explains how to migrate data to and from INFORMIX-OnLine/Secure systems.

The Appendix portions of the SFUG contain information about 4GLs and Error Messages, Glossary information, and an Index.

Conclusion

INFORMIX-OnLine/Secure satisfies the C2 Security Features User's Guide requirement.

7.9 Trusted Facility Manual

Requirement

TCSEC:

A manual addressed to the ADP system administrator shall present cautions about functions and privileges that should be controlled when running a secure facility. The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given.

TDI:

This requirement applies as stated in the TCSEC to the TCB and to every TCB subset in the TCB.

This requirement can be met by providing a set of manuals, one for each distinct (non-replicated) TCB subset. Each manual shall address the functions and privileges to be controlled for the associated TCB subset. Additionally, it must clearly show the interfaces to, and the interaction with, more primitive TCB subsets. The manual for each TCB subset shall identify the functions and privileges of the TCB subsets on which the associated TCB subset depends. Also, the TCB subset's manual shall identify which, if any, configuration options of the more primitive TCB subsets are to be used for the composite TCB to operate securely.

Any pre-defined roles supported (e.g., database administrator) by the TCB subset shall be addressed.

The means for correlating multiple audit logs and synonymous user identifications from multiple TCB subsets (if such exist) shall also be addressed.

The trusted facility manual shall describe the composite TCB so that the interactions among the TCB subsets can be readily determined. Other manuals may be referenced in this determination. The manuals that address the distinct modules of the TCB and the generation of the TCB need to be integrated with the other trusted facility manuals only to the extent that they are affected by the use and operation (versus the development) of the composite TCB.

Applicable Features

The *INFORMIX OnLine/Secure Trusted Facility Manual Database Server Version 4.1* is intended for use by system operators and administrators. This manual is used to guide an administrator or operator in the correct way to operate INFORMIX-OnLine/Secure in a secure manner, and meets the Trusted Facility Manual requirements.

This manual consists of sections which describe the procedures for operating the system in a secure manner. System Roles, Security Overview, Installation and Setup, Operating System Administrator Responsibilities, Database System Security Officer Responsibilities, Database System Administrator Responsibilities, Coordinated Responsiblities, and Audit are all covered in this manual.

Conclusion

INFORMIX-OnLine/Secure satisfies the C2 Trusted Facility Manual requirement.

7.10 Test Documentation

Requirement

TCSEC:

The system developer shall provide to the evaluators a document that describes the test plan, test procedures that show how the security mechanisms were tested, and results of the security mechanisms' functional testing.

TDI:

This requirement applies as stated in the TCSEC to the composite TCB.

Applicable Features

The *Security Test Plan for INFORMIX-OnLine/Secure* describes the general approach to testing, the test methodology, and the test plan. This document also describes both the software and hardware test environment. This is particularly important since INFORMIX-OnLine/Secure runs in the context of a secure operating system.

The INFORMIX-OnLine/Secure Security Test Suite described in the test plan comprises five formal tests which are designed to test the mechanisms used to enforce the security policy. The identified security mechanisms include: User Identification, DAC, and Audit. Sensitivity Labels and MAC are included in the B1 configurations. Each formal test comprises one or more test case. The test plan describes the objective of each test case and the INFORMIX-OnLine/Secure function(s) tested in each test case.

The actual test documentation is found in *Security Test Procedures for INFORMIX-OnLine/Secure.* This document contains a more detailed description of each test case and includes requirements traceability which identifies all INFORMIX-OnLine/Secure functions tested for each test case and maps each function to the Verification Cross Reference Matrix (VCRM) found in the test plan. The VCRM in turn maps each function being tested to a particular security requirement. Documentation is provided which identifies where additional information can be found about the design of the function which implements each security requirement. In this way, Informix provides coverage of the security-relevent functions of INFORMIX-OnLine/Secure.

There are approximately 30 test procedures included in the appendix to this document. Each procedure is kept in a separate folder and includes a step by step description of how to conduct the test, the expected results for each step, and a copy of the terminal displays (for cases where the displays provide additional information). Initialization procedures are also provided for each procedure, where applicable. Although the majority of the tests are manual, and conducting the tests is a lengthy process, the available test documentation is exceptional and easy to understand.

Conclusion

INFORMIX-OnLine/Secure satisfies the C2 Test Documentation requirement.

7.11 Design Documentation

Requirement

TCSEC:

Documentation shall be available that provides a description of the manufacturer's philosophy of protection and an explanation of how this philosophy is translated into the TCB. If the TCB is composed of distinct modules, the interfaces between these modules shall be described.

TDI:

This requirement applies as stated in the TCSEC to the composite TCB. If the TCB is composed of multiple subsets, this requirement applies to each TCB subset and the interfaces between TCB subsets.

Applicable Features

Informix provides the *Philosophy of Protection for INFORMIX-OnLine/Secure* which describes the security policy and general security mechanisms of INFORMIX-OnLine/Secure. Informix provides a number of documents which describe how their philosophy of protection is translated into INFORMIX-OnLine/Secure. Informix provides the *System/Segment Design for INFORMIX-OnLine/Secure* and the *System/Segment Specification for Informix* as architecture overview documents.

Informix describes the INFORMIX-OnLine/Secure TCB subset as comprising two main components: the RDBMS Kernel and the Secure Administrator's Front End. For each of these components, Informix provides the following design documents which discuss the details of the component's implementation and interface to other components:

- Software Requirement Specification
- Software Design Document
- Interface Requirement Specification
- Interface Design Document

To address how the INFORMIX-OnLine/Secure TCB subset interfaces with the operating system subsets in the evaluated configuration, Informix provides interface requirements and specification documents for each of the operating systems. Because INFORMIX-OnLine/Secure requires specific privileges from the operating system to execute, Informix provides information concerning privileges that are required for INFORMIX-OnLine/Secure to execute. For each operating system in the evaluated configuration, Informix provides the following:

- an explicit list of privileges that are provided by the OS and used by INFORMIX-OnLine/Secure

- an explanation of why the identified privileges are needed and how they are used,
- an explanation of what it means to use these privileges in the context of the operating system,
- a supporting statement that using the privileges will not adversely effect the security mechanisms of the underlying operating system.

This information is provided for each operating system in the evaluated configuration.

Conclusion

INFORMIX-OnLine/Secure satisfies the C2 Design Documentation requirement.

This page intentionally left blank

Chapter 8

Evaluation as a B1 System

This chapter describes how the TCSEC requirements and the TDI interpretations are satisfied by INFORMIX-OnLine/Secure executing on top of operating systems in the evaluated configuration. The rating earned by Informix will be associated with the composite TCB comprising INFORMIX-OnLine/Secure and either AT&T System V/MLS, Harris CX/SX, Harris CX/SX with LAN/SX, or HP-UX BLS. Consequently, the requirements are considered from the perspective of the composite TCB. As described throughout this report, INFORMIX-OnLine/Secure depends heavily on the operating system security mechanisms to enforce its portion of the composite security policy; therefore the explanation of how some requirements are met lies in how the evaluated operating system meets the requirements. This chapter does not address the details of how the evaluated operating system meets the TCSEC requirements, but addresses what INFORMIX-OnLine/Secure explicitly relies on to enforce its portion of the policy. For details concerning each operating system design and implementation, see each of the Final Evaluation Reports [[5], [6], [7].

8.1 Discretionary Access Control

Requirement

TCSEC:

The TCB shall define and control access between named users and named objects (e.g., files and programs) in the ADP system. The enforcement mechanism (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of those objects by named individuals, or defined groups of individuals, or by both, and shall provide controls to limit propagation of access rights. The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorized access. These access controls shall be capable of including or excluding access to the granularity of a single user. Access permission to an object by users not already possessing access permission shall only be assigned by authorized users.

TDI:

The discretionary access control requirements apply as stated in the TCSEC to every TCB subset whose policy includes discretionary access control of its subjects to its objects. Any TCB subset whose policy does not include such discretionary access control is exempt from this requirement.[1]

[1] Note that any evaluation by parts requires that at least one TCB subset in the TCB enforce a discretionary access control policy, and thus satisfy this requirement.

Applicable Features

The OS provides DAC on named objects under its control such as files and devices. The INFORMIX-OnLine/Secure DAC mechanism is distinct from that of the OS; it replaces the OS DAC policy within the RDBMS by applying access attributes specific to RDBMS named objects, rather than just the read, write, and execute control placed on OS named objects. The RDBMS named objects correspond to databases, tables, views, synonyms, constraints, and indexes. DAC is accomplished via privileges which users grant and revoke using SQL statements or the RSAM interface. Privileges are granted to single users by name or to all users under the name of PUBLIC.

When a new named object is created, the *initial access* to it is established by default. If the named object is a database, the user who creates the database is given the **dba** privilege. If the named object is a table, the creating user becomes the owner and is given all table level privileges including the permission to grant and revoke these privileges to and from other users. If the named object is a view, synonym, constraint, or index, the creating user becomes the owner of the object but gets no specific privileges since none are defined for these types of named objects. No user other than the creator initially possesses any privileges to a named object with the exception of the Database Administrator (called the dba) who has implicit privileges on the tables in the database.

Changes in the discretionary access of a user to a named object can occur when a subject explicitly grants or revokes privileges to or from the user to the named object. A subject cannot grant or revoke a database level privilege (**dba**, **resource**, **connect**) to or from another user unless the subject possesses the **dba** privilege. A user, other than the table owner, possesses a table level privilege only if someone else previously granted that user the privilege. Each table level privilege can have a *grant* option associated with it which allows the user to give the privilege to another user.

A user can only revoke a table level privilege from another user if the revoking user originally granted the table level privilege. However, table level privileges can be indirectly revoked from a user through a chain of cascading effects. In addition, INFORMIX-OnLine/Secure prevents cycles of grants. In this way, INFORMIX-OnLine/Secure limits the propogation of access rights.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Discretionary Access Control requirement.

8.2 Object Reuse

Requirement

TCSEC:

All authorizations to the information contained within a storage object shall be revoked prior to initial assignment, allocation or reallocation to a subject from the TCB's pool of unused storage objects. No information, including encrypted representations of information, produced by a prior subject's actions is to be available to any subject that obtains access to an object that has been released back to the system.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset in the TCB.

Applicable Features

INFORMIX-OnLine/Secure doesn't provide any interfaces to an object's resources until the object has been used. An object cannot be read by a user until it has been written into. Resources allocated for an object's use cannot be accessed. The OS enforces an object reuse policy on its objects. See page 82, "Object Reuse" for details regarding object reuse for each object.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Object Reuse requirement.

8.3 Labels

Requirement

TCSEC:

Sensitivity labels associated with each subject and storage object under its control (e.g., process, file, segment, device) shall be maintained by the TCB. These labels shall be used as the basis for mandatory access control decisions. In order to import non-labeled data, the TCB shall request and receive from an authorized user the security level of the data, and all such actions shall be auditable by the TCB.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset whose policy includes mandatory access control of its subjects to its objects. Any TCB subset whose policy does not include such mandatory access control is exempt from this requirement.

Applicable Features

INFORMIX-OnLine/Secure maintains a sensitivity label for subjects and objects in the RDBMS; these sensitivity labels are created by the OS administrator for the RDBMS. The sensitivity labels are used as a basis for all MAC decisions. INFORMIX-OnLine/Secure does not allow the importation of non-labeled data. The only type of data that may be imported is INFORMIX-OnLine/Secure data, which is always labeled. The OS TCB is responsible for maintaining labels associated with the subjects and objects it controls. These OS subjects are a superset of the RDBMS subjects; the subjects with the MAC label **ix_users** are permitted to access the RDBMS. All RDBMS objects have the database category **ix_data** to protect them from unauthorized access.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Labels requirement.

8.4 Label Integrity

Requirement

TCSEC:

Sensitivity labels shall accurately represent security levels of the specific subjects or objects with which they are associated. When exported by the TCB, sensitivity labels shall accurately and unambiguously represent the internal labels and shall be associated with the information being exported.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset whose policy includes mandatory access control of its subjects to its objects. Any TCB subset whose policy does not include such mandatory access control is exempt from this requirement.

Applicable Features

The OS maintains a sensitivity label for each subject and object under its control which represents the security level of the specific subject or object as determined by INFORMIX-OnLine/Secure. Subjects under the control of the DBMS are those OS subjects in the group **ix_users** and objects under the DBMS control are those created by DBMS subjects. When exporting data, the DBSA uses the trusted processes *tbtape* and *tbunload* to write the data. Before the tape is unloaded, the DBSA must request a copy of the labels file from the OS. That labels file is stored with the tape. When the DBSA is ready to load the tape, the OSA from the target machine provides the labels file from the target machine. The DBSA then takes the two lables files and creates the label.map file. This mapping preserves the label representation associated with the data on the tape. It is the responsibility of the OS to maintain label integrity on the data it exports.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Label Integrity requirement.

8.5 Exportation of Labeled Information

Requirement

TCSEC:

The TCB shall designate each communication channel and I/O device as either single-level or multilevel. Any change in this designation shall be done manually and shall be auditable by the TCB. The TCB shall maintain and be able to audit any change in the current security level or levels associated with a communication channel or I/O device.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset whose policy includes mandatory access control of its subjects to its objects. Any TCB subset whose policy does not include such mandatory access control is exempt from this requirement.

Applicable Features

INFORMIX-OnLine/Secure does not provide any communication channels of its own and does not use the OS communication channels. Therefore, this requirement does not apply.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Exportation of Labeled Information requirement.

8.6 Exportation to Multilevel Devices

Requirement

TCSEC:

When the TCB exports an object to a multilevel I/O device, the sensitivity label associated with that object shall also be exported and shall reside on the same physical medium as the exported information and shall be in the same form (i.e., machine-readable or human-readable form). When the TCB exports or imports an object over a multilevel communication channel, the protocol used on that channel shall provide for the unambiguous pairing between the sensitivity labels and the associated information that is sent or received.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset whose policy includes mandatory access control of its subjects to its objects. Any TCB subset whose policy does not include such mandatory access control is exempt from this requirement.

Applicable Features

To the OS, INFORMIX-OnLine/Secure is a single-level process, and its memory space has a single sensitivity label. However INFORMIX-OnLine/Secure uses this space in the OS's filesystem as a repository of multi-level data; thus, from INFORMIX-OnLine/Secure 's point of view this space is a multi-level device. The sensitivity label of the data is stored in the OS files along with the data. The same type of reasoning follows for tapes. Sensitivity labels are preserved across tapes by using the *stbunload* and the *stbtape* processes. The *stbunload* process writes all the sensitivity labels used in the tables to the tape along with each sensitivity label's tag so that they can be properly mapped as described in the *TFM* when the information is reloaded. The *stbtape* process writes all archiving data to tape along with the data's sensitivity labels.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Exportation to Multilevel Devices requirement.

8.7 Exportation to Single-Level Devices

Requirement

TCSEC:

Single-level I/O devices and single-level communication channels are not required to maintain the sensitivity labels of the information they process. However, the TCB shall include a mechanism by which the TCB and an authorized user can reliably communicate to designate the single security level of information imported or exported via single-level communication channels or I/O devices.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset whose policy includes mandatory access control of its subjects to its objects. Any TCB subset whose policy does not include such mandatory access control is exempt from this requirement.

Applicable Features

As described in the previous requirement for multi-level devices, the OS views INFORMIX-OnLine/Secure as a single-level process, but INFORMIX-OnLine/Secure views itself as the handler of multi-level data. For this reason, there are no single-level devices with respect to the composite TCB. The single-level devices INFORMIX-OnLine/Secure does use are provided for and managed by the OS. Hence, this requirement does not apply to INFORMIX-OnLine/Secure.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Exportation to Single-Level Devices requirement.

8.8 Labeling Human-Readable Output

Requirement

TCSEC:

The ADP system administrator shall be able to specify the printable label names associated with exported sensitivity labels. The TCB shall mark the beginning and end of all human-readable, paged, hardcopy output (e.g., line printer output) with human-readable sensitivity labels that properly [2] represent the sensitivity of the output. The TCB shall, by default, mark the top and bottom of each page of human-readable, paged, hardcopy output (e.g., line printer output) with human-readable sensitivity labels that properly represent the overall sensitivity of the output or that properly represent the sensitivity of the information on the page. The TCB shall, by default and in an appropriate manner, mark other forms of human-readable output (e.g., maps, graphics) with human-readable sensitivity labels that properly represent the sensitivity of the output. Any override of these marking defaults shall be auditable by the TCB.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset whose policy includes mandatory access control of its subjects to its objects. Any TCB subset whose policy does not include such mandatory access control is exempt from this requirement.

Applicable Features

INFORMIX-OnLine/Secure relies on the OS to provide printing services. When a user process wants to send data to a printer, the process must make a call to the OS. The OS then prints the data giving it the sensitivity label of the process. For this reason, this requirement does not apply to INFORMIX-OnLine/Secure.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Labeling Human-Readable Output requirement.

8.9 Mandatory Access Control

Requirement

TCSEC:

The TCB shall enforce a mandatory access control policy over all subjects and storage objects under its control (e.g., processes, files, segments, devices). These subjects and objects shall be assigned sensitivity

[2] The hierarchical classification component in human-readable sensitivity labels shall be equal to the greatest hierarchical classification of any of the information in the output that the labels refer to; the non-hierarchical category component shall include all of the non-hierarchical categories of the information in the output the labels refer to, but no other non-hierarchical categories.

labels that are a combination of hierarchical classification levels and non-hierarchical categories, and the labels shall be used as the basis for mandatory access control decisions. The TCB shall be able to support two or more such security levels. The following requirements shall hold for all accesses between subjects and objects controlled by the TCB: A subject can read an object only if the hierarchical classification in the subject's security level is greater than or equal to the hierarchical classification in the object's security level and the non-hierarchical categories in the subject's security level include all the non-hierarchical categories in the object's security level. A subject can write an object only if the hierarchical classification in the subject's security level is less than or equal to the hierarchical classification in the object's security level and all the non-hierarchical categories in the subject's security level are included in the non-hierarchical categories in the object's security level. Identification and authentication data shall be used by the TCB to authenticate the user's identity and to ensure that the security level and authorization of subjects external to the TCB that may be created to act on the behalf of the individual user are dominated by the clearance and authorization of that user.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset whose policy includes mandatory access control of its subjects to its objects. Any TCB subset whose policy does not include such mandatory access control is exempt from this requirement.

Applicable Features

INFORMIX-OnLine/Secure enforces a MAC policy that controls the access of subjects to RDBMS storage objects. All subjects and objects controlled by INFORMIX-OnLine/Secure have a sensitivity label associated with them. The sensitivity label is comprised of a hierarchical classification level and a set of non-hierarchical categories. The meaning of sensitivity labels is dependent on the OS and each OS must support at least two sensitivity labels. The sensitivity labels are used as a basis for MAC decisions. The security policy model for INFORMIX-OnLine/Secure is based on the Bell-LaPadula Model and conforms to the TCSEC policy. Identification and authentication data received from the OS is used to assign authorizations to users. INFORMIX-OnLine/Secure uses OS services to perform sensitivity label comparisons, however INFORMIX-OnLine/Secure manages access to the sensitivity labels of its subjects and objects.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Mandatory Access Control requirement.

8.10 Identification and Authentication

Requirement

TCSEC:

The TCB shall require users to identify themselves to it before beginning to perform any other actions that the TCB is expected to mediate. Furthermore, the TCB shall maintain authentication data that includes information for verifying the identity of individual users (e.g., passwords) as well as information for determining the clearance and authorizations of individual users. This data shall be used by the TCB to authenticate the user's identity and to ensure that the security level and authorizations of subjects external to the TCB that may be created to act on behalf of the individual user are dominated by the clearance and authorization of that user. The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user. The TCB shall be able to enforce individual accountability by providing the capability to uniquely identify each individual ADP system user. The TCB shall also provide the capability of associating this identity with all auditable actions taken by that individual.

TDI:

This requirement applies as stated in the TCSEC to the entire TCB. The cooperative action of the TCB subsets making up the TCB must satisfy the requirement.

If the TCB is composed of TCB subsets, one TCB subset may either rely on a mechanism in another TCB subset to provide identification and authentication services or provide the services directly. Similarly, that TCB subset may rely on a mechanism in another more primitive TCB subset to ensure that the security level of subjects external to the TCB that may be created to act on behalf of the individual user are dominated by the clearance and authorization of that user. Each TCB subset may maintain its own identification and authentication data or one central repository may be maintained. If each TCB subset has its own data, then the information for each individual user must be consistent among all subsets.

Applicable Features

The OS in the evaluated configuration require users to explicitly identify themselves to the TCB with a unique user identifier and to authenticate themselves with passwords before they may access TCB protected resources. Each OS maintains clearance and authorization information for each individual to ensure that users may not invoke or access any data which is protected by labels outside their clearance range. The OS protects this identification and authentication information, making it accessible by OS administrative personnel only. Each audit record contains a unique user ID which is assigned to a user's process at login and is recorded in the audit records generated by all processes invoked by that user.

INFORMIX-OnLine/Secure relies solely on the underlying operating system to provide identification and authentication services; therefore identification and authentication information for the database and the operating system is kept in a central repository protected by the operating system. Although INFORMIX-OnLine/Secure generates its own audit records, it relies on operating system services to provide user identification information to uniquely associate auditable events with individuals. To ensure that user's may not invoke or access information outside their access range, INFORMIX-OnLine/Secure relies on the operating system to properly restrict users from logging into levels or groups to which they do not have access. Specifically, untrusted users are not allowed to have the **IX_DATA** category in their clearance range nor are they allowed to belong to the group **ix_data**.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Identification and Authentication requirement.

8.11 Audit

Requirement

TCSEC:

The TCB shall be able to create, maintain, and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorized for audit data. The TCB shall be able to record the following types of events: use of identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. The TCB shall also be able to audit any override of human-readable output markings. For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the object and the object's security level. The ADP system administrator shall be able to selectively audit the actions of any one or more users based on individual identity and/or object security level.

TDI:

This requirement applies as stated in the TCSEC to the entire TCB. The cooperative action of the TCB subset making up the TCB must satisfy the requirement.

A TCB subset may maintain its own security audit log, distinct from that maintained by more primitive TCB subsets, or it may use an audit interface provided by a different TCB subset allowing the audit records it generates to be processed by that TCB subset.

If the TCB subset uses different user identifications that a more primitive TCB subset, there shall be a means to associate audit records generated by different TCB subsets for the same individual with each other, either at the time they are generated or later.

Auditable events, in the case of a database management system, are the individual operations initiated by untrusted users (e.g., updates, retrievals, and inserts) not just the invocation of the database management system. The auditing mechanism shall have the capability of auditing all mediated accesses which are visible to users. That is, each discretionary access control decision and each mandatory access control policy decision shall be auditable. Individual operations performed by trusted software, if totally transparent to the user, need not be auditable. If the total audit requirement is met by the use of more than one audit log, a method of correlation must be available.

Applicable Features

The RDBMS Kernel creates audit records of events and sends the records to the OS audit log using the interface provided by the OS. The RDBMS Kernel relies on the protection of the OS audit log for protection of RDBMS audit records. The RDBMS Kernel also relies on the OS audit mechanism implementation to limit the number of audit records that can be lost during system failures. The audit records stored in the OS audit log can be extracted to a file that only contains RDBMS Kernel audit records. This file is protected so that only the DBSSO or AAO can access the audit data.

The RDBMS Kernel has the ability to audit object creations and deletions, object accesses, object updates, invocation of the RDBMS Kernel, the use of locks, and the granting and revoking of DAC privileges. DBSA actions can also be audited including the use of support processes and changes to the RDBMS Kernel configuration. DBSSO actions such as maintenance of audit masks, maintenance of DAC privileges, and maintenance of MAC labels are always audited. The OS audits identification and authentication and all other OS specific events.

The information recorded by the RDBMS Kernel for each event includes the real user ID of the process that performed the event, the process's sensitivity label, success or failure of the event, and the event code. The OS audit mechanism supplies an audit header that contains the date and time of the event. An object's name and sensitivity label are included in events that create, delete, access, update, or in any way relate to an object.

The DBSSO can selectively audit on a per user basis using audit masks. A system wide default of auditable events can also be set up. The DBSSO must use the utility *datextract* in order to choose audited events based on an object's sensitivity label. See page 85, "Auditing" for details on the audit mechanism.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Audit requirement.

8.12 System Architecture

Requirement

TCSEC:

The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g., by modification of its code or data structures). Resources controlled by the TCB may be a defined subset of the subjects and objects in the ADP system. The TCB shall maintain process isolation through the provision of distinct address spaces under its control. The TCB shall isolate the resources to be protected so that they are subject to the access control and auditing requirements.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset, with the following additional interpretations.

The TCB must provide domains for execution that are allocated to and used by TCB subsets according to the subset-domain condition for evaluation by parts. A most primitive TCB subset must provide domains for execution. A less primitive TCB subset must make use of domains provided by a more primitive TCB subset. A less primitive TCB subset may provide further execution domains but is not required to do so.

Similarily, the TCB must provide distinct address spaces for untrusted processes. A most primitive TCB subset must provide address spaces for its subjects. A less primitive TCB subset must make use of distinct address space provided by a more primitive TCB subset. A less primitive TCB subset may provide more fine-grained distinct address spaces, but is not required to do so.

If the TCB is composed of multiple TCB subsets, this requirement [for protection from tampering] applies to each TCB subset.

Applicable Features

The OS TCB subset implements a domain for its execution that protects it from external interference or tampering as well as process isolation through distinct process address spaces. It protects resources it controls and subject those resources to access control and auditing requirements. Details of the each of the operating systems' domain and process isolation mechanisms can be found in the Final Evaluation Report for each operating system [[5], [6], [7]].

The INFORMIX-OnLine/Secure TCB subset relies on OS mechanisms and services to maintain its domain isolation and protect it from external interference or tampering. Specifically, INFORMIX-OnLine/Secure executes as a trusted process in the context of the operating system and uses the OS's process isolation properties to maintain and protect its execution domain. INFORMIX-OnLine/Secure uses the OS's MAC and DAC mechanisms to isolate its data structures (i.e., global shared memory and disk data structures) and subjects the resources it controls to access control and auditing policies. Specifically, it stores RDBMS information in OS objects protected by a special MAC sensitivity label (**Datahi+IX_DATA**) which no untrusted users have in their clearance range. Each OS object containing RDBMS information is protected by traditional UNIX protection bits with access granted only to members of the **ix_data** group. INFORMIX-OnLine/Secure relies upon the OS process isolation coupled with the proper management of its resources, as described in Chapter 3, to provide each RDBMS user a distinct address space. All INFORMIX-OnLine/Secure executables are protected from tampering using traditional UNIX protection bits with execute only permission granted to RDBMS users.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 System Architecture requirement.

8.13 System Integrity

Requirement

TCSEC:

Hardware and/or software features shall be provided that can be used to periodically validate the correct operation of the on-site hardware and firmware elements of the TCB.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset that includes hardware or firmware. Any TCB subset that does not include hardware or firmware is exempt from this requirement.

Applicable Features

This requirement is not applicable to INFORMIX-OnLine/Secure and is handled completely in the OS.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 System Integrity requirement.

8.14 Security Testing

Requirement

TCSEC:

The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. A team of individuals who thoroughly understand the specific implementation of the TCB shall subject its design documentation, source code, and object code to thorough analysis and testing. Their objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users. All discovered flaws shall be removed or neutralized and the TCB retested to demonstrate that they have been eliminated and that new flaws have not been introduced.

TDI:

This requirement applies as stated in the TCSEC to the entire TCB. If a TCB consists of TCB subsets meeting the conditions for evaluation by parts, the satisfaction of the requirements by each TCB subset satisfies the requirement for the entire TCB. Otherwise, security testing of the entire TCB must be performed (even if the results of testing the individual TCB subsets were available).

Applicable Features

In addition to a test readiness review, the team verified the test tools that were used to run the test suite to ensure that they were operating properly. This included verifying the SAFE viewing mechanism and audit record generation based on a single audit flag.

The evaluation team observed and participated in the manual execution of the Informix test suite on all three configurations: C2, B1/EA, and B1/EP. Although this task was tedious, the procedures were easy to follow and the results were as expected. Several inconsistencies were uncovered during the first week of testing but were re-tested and resolved during the second week of testing.

Of particular concern was the fact that all testing was conducted at the interface to the SQL Engine. Since the RSAM interface is also part of the TCB interface in the B1/EA configuration, it was necessary to ensure that all RSAM interfaces were covered by the tests. To do this, all tests were run with a slightly modified

RSAM that dumped all RSAM interface calls (in the form of RSAM messages) to a file. The team was able to examine this file to ensure complete coverage of the RSAM interface. Some RSAM interfaces, however, cannot be invoked directly via SQL commands. To test these interfaces, Informix provided a special menu-driven tool that allowed the tester to chose the interface to be tested and subsequently performed both a success and failure test. After each test was run, the audit log was examined and all test results were as expected. The team did not perform testing of the OS portion of the TCB. Instead, the team performed analysis of the RDBMS-OS interface and conducted tests to support the assertion that INFORMIX-OnLine/Secure does not inappropriately modify or corrupt the OS portion of the TCB. The team chose to perform more in-depth analysis at the RDMBS-OS interface rather than test the entire TCB because the team felt this approach provided more assurance. To conduct testing of the interface, the team identified security relevant OS files to monitor throughout the testing exercise. These files were backed up to tape and periodically during testing, the current state of these files was compared to the backup copy. These files were checked on each of the three testing machines with no inconsistencies found. For each machine, the team also periodically reviewed the OS audit trail for suspicious behavior by INFORMIX-OnLine/Secure. Specifically, the team looked for unexplained object reads and writes from test users on whose behalf INFORMIX-OnLine/Secure was executing. No unexplained actions were uncovered. A number of team tests were also run including passing invalid messages to RSAM; attempting to access global shared memory from an unprivileged user account; object reuse tests such as attempting to access a row that has been deleted and attempting to read rows from an empty table; and attempting to perform normal user activities while logged in as the DBSSO or DBSA. Several other team tests were also performed and all results were as expected.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Security Testing requirement.

8.15 Design Specification and Verification

Requirement

TCSEC:

A formal or informal model of the security policy supported by the TCB shall be maintained over the life cycle of the ADP system and demonstrated to be consistent with its axioms.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset, with the following specific additional interpretations.

It must be demonstrated that the security policy enforced by the composite TCB is represented by the collection of policy models for the individual TCB subsets.

If the TCB is composed of multiple TCB subsets, this requirement applies to the security policy of each TCB subset. An informal argument that the set of policy models represents the “security policy supported by the [composite] TCB” must be provided.

Applicable Features

Informix has produced an informal model that is consistent with its axioms. The model addresses the security policy required by the OS , describes the security policy INFORMIX-OnLine/Secure enforces between its subjects and objects, and addresses the security policy enforced by the combination of the OS and INFORMIX-OnLine/Secure. See page 94, "Model" for more detail.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Design Specification and Verification requirement.

8.16 Security Features User's Guide

Requirement

TCSEC:

A single summary, chapter, or manual in user documentation shall describe the protection mechanisms provided by the TCB, guidelines on their use, and how they interact with one another.

TDI:

This requirement applies as stated in the TCSEC to every TCB subset in the TCB. This collection of guides must include descriptions of every TCB subset in the TCB and explicit cross-references to other related user's guides to other TCB subsets, as required. In addition, interactions between mechanisms within different TCB subsets must be clearly described.

Applicable Features

The *INFORMIX OnLine/Secure Security Features User's Guide Database Server Version 4.1* is intended for users and provides descriptions of the functions of INFORMIX-OnLine/Secure.

The preface and the introduction of this document describes the contents of the manual and the proper use of the manual. In addition, these sections point to other reference materials that will help the user understand general security concepts. Finally, the introduction describes the different user roles and system configurations.

Chapter 1 of the SFUG is entitled "Security Concepts" and provides an overview of the security mechanisms utilized by INFORMIX-OnLine/Secure, as well as some security terminology.

Chapter 2, "Using INFORMIX-OnLine/Secure in a Multilevel Environment," describes how B1 INFORMIX-OnLine/Secure server is different from the general comercial product. It includes topics like multilevel tables, locking in a multilevel environment, how SQL statements behave in a multilevel environment, syntax and use of label comparison functions, and assorted session configuration information.

Chapter 3, "Using INFORMIX-OnLine/Secure in a Single-Level Environment," explains how a C2 INFORMIX-OnLine/Secure server is different from the comercial product. It discusses disk management of single-level tables.

Chapter 4, "Using INFORMIX-OnLine/Secure in Any Environment," contains information that users of either a C2 or B1 INFORMIX-OnLine/Secure server should know. It covers how some of the privilege-related functions differ from INFORMIX-OnLine/Secure, effective checking on unique constraints, parallel sorting for multiprocessor machines, and how utilities function.

Chapter 5, "Import/Export," explains how to migrate data to and from INFORMIX-OnLine/Secure systems.

The Appendix portions of the SFUG contain information about 4GLs and Error Messages, Glossary information, and an Index.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Security Features User's Guide requirement.

8.17 Trusted Facility Manual

Requirement

TCSEC:

A manual addressed to the ADP system administrator shall present cautions about functions and privileges that should be controlled when running a secure facility. The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given. The manual shall describe the operator and administrator functions related to security, to include changing the security characteristics of a user. It shall provide guidelines on the consistent and effective use of the protection features of the system, how they interact, how to securely generate a new TCB, and facility procedures, warnings, and privileges that need to be controlled in order to operate the facility in a secure manner.

TDI:

This requirement applies as stated in the TCSEC to the TCB and to every TCB subset in the TCB.

This requirement can be met by providing a set of manuals, one for each distinct (non-replicated) TCB subset. Each manual shall address the functions and privileges to be controlled for the associated TCB subset. Additionally, it must clearly show the interfaces to, and the interaction with, more primitive TCB subsets. The manual for each TCB subset shall identify the functions and privileges of the TCB subsets on which the associated TCB subset depends. Also, the TCB subset's manual shall identify which, if any, configuration options of the more primitive TCB subsets are to be used for the composite TCB to operate securely.

Any pre-defined roles supported (e.g., database administrator) by the TCB subset shall be addressed.

The means for correlating multiple audit logs and synonymous user identifications from multiple TCB subsets (if such exist) shall also be addressed.

The trusted facility manual shall describe the composite TCB so that the interactions among the TCB subsets can be readily determined. Other manuals may be referenced in this determination. The manuals that address the distinct modules of the TCB and the generation of the TCB need to be integrated with the other trusted facility manuals only to the extent that they are affected by the use and operation (versus the development) of the composite TCB.

Applicable Features

The *INFORMIX OnLine/Secure Trusted Facility Manual Database Server Version 4.1* is intended for use by system operators and administrators. This manual is used to guide an administrator or operator in the correct way to operate INFORMIX-OnLine/Secure in a secure manner, and meets the Trusted Facility Manual requirements.

This manual consists of sections which describe the procedures for operating the system in a secure manner. System Roles, Security Overview, Installation and Setup, Operating System Administrator Responsibilities, Database System Security Officer Responsibilities, Database System Administrator Responsibilities, Coordinated Responsiblities, and Audit are all covered in this manual.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Trusted Facility Manual requirement.

8.18 Test Documentation

Requirement

TCSEC:

The system developer shall provide to the evaluators a document that describes the test plan, test procedures that show how the security mechanisms were tested, and results of the security mechanisms' functional testing.

TDI:

This requirement applies as stated in the TCSEC to the composite TCB.

Applicable Features

The *Security Test Plan for INFORMIX-OnLine/Secure* describes the general approach to testing, the test methodology, and the test plan. This document also describes both the software and hardware test environment. This is particularly important since INFORMIX-OnLine/Secure runs in the context of a secure operating system.

The INFORMIX-OnLine/Secure Security Test Suite described in the test plan comprises five formal tests which are designed to test the mechanisms used to enforce the security policy. The identified security mechanisms include: User Identification, DAC, and Audit. Sensitivity Labels and MAC are included in the B1 configurations. Each formal test comprises one or more test case. The test plan describes the objective of each test case and the INFORMIX-OnLine/Secure function(s) tested in each test case.

The actual test documentation is found in *Security Test Procedures for INFORMIX-OnLine/Secure*. This document contains a more detailed description of each test case and includes requirements traceability which identifies all INFORMIX-OnLine/Secure functions tested for each test case and maps each function to the Verification Cross Reference Matrix (VCRM) found in the test plan. The VCRM in turn maps each function being tested to a particular security requirement. Documentation is provided which identifies where additional information can be found about the design of the function which implements each security requirement. In this way, Informix provides coverage of the security-relevent functions of INFORMIX-OnLine/Secure.

There are approximately 30 test procedures included in the appendix to this document. Each procedure is kept in a separate folder and includes a step by step description of how to conduct the test, the expected results for each step, and a copy of the terminal displays (for cases where the displays provide additional information). Initialization procedures are also provided for each procedure, where applicable. Although the majority of the tests are manual, and conducting the tests is a lengthy process, the available test documentation is exceptional and easy to understand.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Test Documentation requirement.

8.19 Design Documentation

Requirement

TCSEC:

Documentation shall be available that provides a description of the manufacturer's philosophy of protection and an explanation of how this philosophy is translated into the TCB. If the TCB is composed of distinct modules, the interfaces between these modules shall be described. An informal or formal description of the security policy model enforced by the TCB shall be available and an explanation provided to show that it is sufficient to enforce the security policy. The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model.

TDI:

This requirement applies as stated in the TCSEC to the composite TCB. If the TCB is composed of multiple subsets, this requirement [for module interfaces] applies to each TCB subset and the interfaces between

TCB subsets. If the TCB is composed of multiple subsets, this requirement [for identification of protection mechanisms] applies to each TCB subset and shall include the protection mechanisms which support the conditions for TCB subset structure and separate subset domains.

Applicable Features

Informix provides the *Philosophy of Protection for INFORMIX-OnLine/Secure* which describes the security policy and general security mechanisms of INFORMIX-OnLine/Secure. They provide the *Multilevel Data Model for INFORMIX-OnLine/Secure* and the *Security Policy Model* which combine to model INFORMIX-OnLine/Secure subjects and objects and how the policies supported by INFORMIX-OnLine/Secure integrate into the overall policy enforced by the operating system.

Informix provides a number of documents which describe how their philosophy of protection and model are translated into INFORMIX-OnLine/Secure. Informix provides the *System/Segment Design for INFORMIX-OnLine/Secure* and the *System/Segment Specification for Informix* as architecture overview documents.

Informix describes the INFORMIX-OnLine/Secure TCB subset as comprising two main components: the RDBMS Kernel and the Secure Administrator's Front End. For each of these components, Informix provides the following design documents which discuss the details of the component's implementation and interface to other components:

- Software Requirement Specification
- Software Design Document
- Interface Requirement Specification
- Interface Design Document

To address how the INFORMIX-OnLine/Secure TCB subset interfaces with the operating system subsets in the evaluated configuration, Informix provides interface requirements and specification documents for each of the operating systems. Because INFORMIX-OnLine/Secure requires specific privileges from the operating system to execute, Informix provides information concerning privileges that are required for INFORMIX-OnLine/Secure to execute. For each operating system in the evaluated configuration, Informix provides the following:

- an explicit list of privileges that are provided by the OS and used by INFORMIX-OnLine/Secure
- an explanation of why the identified privileges are needed and how they are used,
- an explanation of what it means to use these privileges in the context of the operating system,
- a supporting statement that using the privileges will not adversely effect the security mechanisms of the underlying operating system.

This information is provided for each operating system in the evaluated configuration.

Conclusion

INFORMIX-OnLine/Secure satisfies the B1 Design Documentation requirement.

This page intentionally left blank

Chapter 9

Evaluators' Comments

9.1 Invisible Locks

Invisible locks allow a process with a higher sensitivity label to lock an object with a lower sensitivity label. The process at the lower sensitivity label can still access the object for both read and write. However, the process that originated the lock is notified of any changes made to the object. In this case, the higher sensitivity label process can perform a roll back to preserve the semantics of a read operation.

INFORMIX-OnLine/Secure implements the Bell and LaPadula model which allows processes to read information at a lower sensitivity label. The use of invisible locks effectively prevents a process at a lower sensitivity label from gaining information about any processes at higher senstivity labels via illicit means. The team feels this locking mechanism is useful in preventing the signaling of information between sensitivity lables.

9.2 Bundles

The use of the bundle abstraction is a unique approach to implementing multi-level tables. Conceptually, the user sees a true multi-level table with each row at a potentially different sensitivity label. In actuality, a multi-level table is a bundle, which is really an index to each single-level table, one for each sensitivity label, in the multi-level table. Although a unique implementation, the bundle abstraction does not prohibit the occurance of polyinstantiated rows. INFORMIX-OnLine/Secure will not cause polyinstantiation to occur, however, it is possible if, for example, a process at a higher sensitivity label reads a row with a lower sensitivity label and subsequently inserts the row back in the table.

9.3 Cascading Effects of Grant and Revoke

A user can only revoke a table level privilege from another user if the revoking user originally granted the table level privilege. However, table level privileges can be indirectly revoked from a user through a chain of cascading effects. For example, suppose a user grants the **Select** privilege with the grant option to a second user, and that user in turn grants the **select** privilege to a third user. When the first user revokes the privilege from the second user, it is also revoked from the third user. This implementation effectively limits the propagation of access rights.

9.4 Manual Test Suite

The execution of the test suite provided by INFORMIX-OnLine/Secure to meet the Test Documentation requirement is entirely manual. This implementation results in a test suite that is tedious to run and maintain.

9.5 Trusted Facility Management

It is important to note that INFORMIX-OnLine/Secure implements trusted facility management by supporting two distinct roles - Database System Security Officer and Database System Administrator. Role-related actions are performed from within a restricted sensitivity label using an administrator front-end. INFORMIX-OnLine/Secure requires the OS administrator to create these specific role sensitivity labels.

9.6 Flexible and Modular Architecture

INFORMIX-OnLine/Secure's architectural design allows it to be configured in a number of ways to meet customer security needs. For customer's who need only Discretionary Access Control, there is the C2 configuration. In this configuration the RSAM and SQL Engine execute together in a single OS process to enforce a DAC security policy on RDMS objects with maximum performance. For customers requiring Mandatory Access Control, INFORMIX-OnLine/Secure can be configured in one of two ways: the B1/EP (Enhanced Performance) and the B1/EA (Enhanced Assurance) configurations. The B1/EP configuration is much like the C2 configuration as the SQL Engine and the RSAM are linked into a single OS process; however, in addition to DAC, it extends the OS's MAC policy to RDBMS objects. The B1/EA configuration extracts the SQL Engine from the TCB. In this configuration, RSAM and the SQL Engine execute as two operating system processes, an untrusted SQL Engine process which communicates with the RSAM process through session shared memory.

Appendix A
Evaluated Components

A.1 Operating Systems

System V/MLS Harris CX/SX, and HP-UX BLS are the operating system systems in the evaluated configurations: These configurations encompass all TCB hardware and software that is included in the operating systems evaluated configurations which are specified in:

1. Final Evaluation Report, American Telephone and Telegraph Company, System V/MLS[5]
2. Final Evaluation Report, Harris Computer Systems, CX/SX and LAN/SX[6]
3. Final Evaluation Report, Hewlett Packard Corporation, HP-UX BLS[7]

A.2 INFORMIX-OnLine/Secure Components

The INFORMIX-OnLine/Secure 4.1 TCB is made up of a number programs which are listed in Table A.1.

datconfig	*datextract*	*rsam*	*safe*	*stbcheck*
stbinit	*stbload*	*tbtape*	*stblog*	*stbmode*
stbparams	*stbspaces*	*stbstat*	*stbtape*	*stbunload*
sqlturbo (EP/C2)	*tbcheck*	*tbinit*	*tbload*	*tblog*
tbmode	*tbparams*	*tbspaces*	*tbstat*	*tbunload*

Table A.1. INFORMIX-OnLine/Secure Components

This page intentionally left blank

Appendix B
Acronyms

AAO	Audit Analysis Officer
AFE	Administrator Front End
BFD	Bundle File Descriptor
BFM	BLOB free map
BLOB	Binary Large Object
CFET	chunk free-extent table
CI	Configuration Item
CRB	Configuration Review Board
DAC	Discretionay Access Control
DAP	Design Analysis Phase
DB	Database
DBA	Database Administrator
DBMS	Database Management System
DBSA	Database System Administrator
DBSSO	Database System Security Officer
EA	Enhanced Assurance
EP	Enhanced Performance
FD	File Descriptor
FER	Final Evaluation Report
IPAR	Initial Product Assessment Report
ISFD	Indexed Sequential File Descriptor
LRU	Least Recently Used
MAC	Mandatory Access Control
OSA	Operating System Administrator
PTR	Preliminary Technical Review
RAMP	Ratings Maintenance Phase
RCO	Responsible Corporate Officer
RDBMS	Relational Database Management System
RSAM	Relational Storage Access Method
SAFE	Secure Administrator Front End
TCB	Trusted Computing Base
TCSEC	Trusted Computer System Evaluation Criteria
TDI	Trusted Database Interpretation
UFE	User Front End
VAP	Vendor Assistance Phase
VCRM	Verification Cross Reference Matrix
VSA	Vendor Security Analyst

This page intentionally left blank

Appendix C

Bibliography and References

[1] Bell, D.E. and LaPadula, L.J., "Secure Computer System: Unified Exposition and Multics Interpretation," MTR-2997 Revision 1, March 1976.

[2] Codd, E.F., "The Relational Model for Database Management," Version 2, Addison-Wesley, 1990.

[3] **Department of Defense Trusted Computer System Evaluation Criteria**, DOD 5200.28-STD, December 1985.

[4] **Explaining the Use of OS Privilege by OnLine/Secure**, December 1993.

[5] **Final Evaluation Report**, American Telephone and Telegraph Company, System V/MLS Release 1.2.0, Running on UNIX System V, Release 3.3.1, Rating Maintenance Plan, CSC-EPL-90/003, September 28, 1990.

[6] **Final Evaluation Report**, Harris Computer Systems, CX/SX Version 6.1 and LAN/SX Version 6.1, TDB.

[7] **Final Evaluation Report**, Hewlett Packard Corporation, HP-UX BLS Version 8.04, TBD.

[8] **Information Technology - Database Language SQL**, X3H2-92-154, American National Standards Institute (ANSI), 1992.

[9] **INFORMIX-OnLine/Secure Security Features User's Guide**, Informix Software, Inc., 000-7160, January 1992.

[10] **INFORMIX-OnLine/Secure Trusted Facility Manual**, Informix Software, Inc., January 1992.

[11] **Interface Design Document for INFORMIX-OnLine/Secure Kernel**, Informix Software, Inc., MLS-B13-0-90, August 1991.

[12] **Interface Design Document for SAFE CSCI INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-B14-0-91, August 1992.

[13] **Interface Requirements Specification for Kernel CSCI INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-B03-1-91, August 1992.

[14] **Interface Requirements Specification for SAFE CSCI INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-B11-1-91, August 1992.

[15] **Multilevel Data Model for INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-E12-0-90, June 1992.

[16] **Philosophy of Protection for INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-E06-0-90, June 1992.

[17] **RM-PLAN for INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-E14-1-92, November 1992.

[18] **SAFE Software Design Document for the INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-B07-0-90, October 1991.

[19] **Security Policy Model**, Informix Software, Inc., MLS-E07-0-90, July 1992.

[20] **Security Test Plan for INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-E014-0-90, July 1992.

[21] **Security Test Procedures for INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-F05-0-91, July 1992.

[22] **Software Design Document for the INFORMIX-OnLine/SecureKernel CSCI**, Informix Software, Inc., MLS-B06-1-90, September 1992.

[23] **Software Requirements Specification for the Kernel INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-B05-0-90, June 1992.

[24] **Software Requirments Specification for the SAFE CSCI of INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-B04-0-90, June 1992.

[25] **System/Segment Design Document for INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-B01-1-90, June 1992.

[26] **System/Segment Specification for INFORMIX-OnLine/Secure**, Informix Software, Inc., MLS-B02-1-90, June 1992.